



EXTENSIBLE EDITOR TECHNICAL MANUAL

Version 2.6
April 1999

Created by the Center for Applied Informatics
Research and Operations Indianapolis, Indiana

And Distributed by
Department of Veterans Affairs
VISTA Technical Services

Table of Contents

INTRODUCTION.....	1
1. RELATED FILES.....	3
1.1 EXTENSIBLE EDITOR FILE	3
1.2 EXTENSIBLE EDITOR MESSAGE FILE.....	7
1.3 EXTENSIBLE EDITOR SPELLCHECKER FILE	82
2. CREATING CONFIGURATIONS.....	9
2.1 INTRODUCTION	9
2.2 DISTRIBUTED CONFIGURATIONS.....	9
2.3 CREATING KEYSTROKE MACROS.....	10
2.4 ESTABLISHING KEY MAPPINGS.....	10
2.5 COMPILING KEY MAPPINGS	11
2.6 THE STATE VARIABLES.....	12
2.7 CREATING IMPORT FILTERS	12
2.8 USING REPLACEABLE PARAMETERS	13
2.9 OTHER CONFIGURATION PARAMETERS	14
3. APPLICATION PROGRAMMING INTERFACE.....	15
3.1 INTRODUCTION	15
3.2 APPLICATION VARIABLES	15
3.3 THE STATE VARIABLE ARRAY.....	18
3.4 TEXT BUFFERS	21
3.5 SUBROUTINE LIBRARY	22
4. UTILITIES.....	123
4.1 RGED	123
4.2 RGEDCMP	123
4.3 RGEDFIL.....	123
4.4 REDIT.....	124
4.5 RGEDKD.....	125
5. ROUTINES.....	127
6. EXPORTED OPTIONS	129
7. ARCHIVING AND PURGING.....	129
8. EXTERNAL RELATIONS.....	129
9. INTERNAL RELATIONS.....	133
10. SECURITY.....	135
11. PACKAGE WIDE VARIABLES.....	137
12. GLOSSARY	141

Table of Contents

INDEX.....	143
ACKNOWLEDGEMENTS	149

Introduction

Full screen editing capability is an important feature of any database management system that deals with a significant amount of free text. Such is certainly the case with **VISTA**, most notably in packages such as MailMan and Radiology where free text comprises the vast bulk of data stored. Yet historically full screen editing in **VISTA** has been limited to the VA FileMan Editor, which is not adequate for more sophisticated editing needs, or the VMS EDT editor (the full screen VMS editor) which has more features, but is not hardware independent. As the Veterans Health Administration (VHA) moves toward the fully electronic medical record, the need to provide full-featured text processing within **VISTA** becomes even more pressing.

Recognizing this reality, we set about designing a sophisticated text editor that would address these needs. As we proceeded with this task, it became apparent that several related needs could be addressed by creating a more generic and configurable product. These needs included a better hypertext help navigator, a text browser for read-only viewing of reports, and the ability to easily create functional extensions. The collective result of all these design considerations was the creation of the Extensible Editor. Modeled after configurable editors like Brief, the Extensible Editor permits the creation of multiple configurations tailored to the specific needs of the application invoking it and defines a detailed application programming interface by which developers may create functional extensions that seamlessly integrate into the editor environment.

Like the VA FileMan Editor, the Extensible Editor is written in the M programming language and is independent of the underlying hardware platform. And like the VA FileMan Editor, it may be defined as an alternate **VISTA** editor or may be invoked through an imbedded function call from within another application. Because the Extensible Editor uses formal parameter passing and makes liberal use of M's NEW command to encapsulate its data, it may even be called recursively. As a practical application of this capability, the Extensible Editor's built-in hypertext help navigator is actually a recursive invocation of the editor using the HELP configuration.

The Extensible Editor provides many editing capabilities not previously available to **VISTA** users. These include key mapping, block selection, cutting and pasting, text search and replace, multiple buffers, user-defined macros, paragraph formatting, and text importing. A specific configuration for the M programmer permits simultaneous editing of multiple M routines and provides extensions that perform useful operations. These include a comment formatter, a general purpose calculator, a routine size calculator, an ASCII code table, and context-sensitive hypertext help that will provide a detailed reference of the M programming language and API calls to Kernel and **VISTA** packages.

Because so much of the behavior of the Extensible Editor is determined by the configuration definition used to invoke it, the editor is readily adaptable to a variety of applications without writing a single line of code. Where additional functionality is required, the ability to create application-specific extensions means the editor can be imparted with an awareness of the invoking application resulting in high level integration. We have used this flexibility to integrate the editor into our clinical decision support system as a protocol authoring tool and into our narrative report system as a text processor and report browser.

1. Related Files

Two files accompany the Extensible Editor. These are the *EXTENSIBLE EDITOR* file containing information on each of the editor's configurations, and the *EXTENSIBLE EDITOR SPELLCHECKER* file, containing the correct spellings of thousands of words. The *EXTENSIBLE EDITOR MESSAGE* file is now obsolete and has been removed from this version. Each of these files will be discussed in detail.

1.1 Extensible Editor File

This is the heart of the Extensible Editor. Within this file, you may define multiple configurations that may be invoked by the editor. Care must be exercised when creating a new entry or modifying an existing one. Either requires a detailed understanding of how this file interacts with the editing environment.

This file is organized by configurations. Each configuration is assigned a name as its primary key (.01 field) and it is this name, passed as a formal parameter, that the editor uses to locate configuration information. Within each configuration are six sections: configuration information, key mappings, key macro definitions, state variable definitions, import filters, and configuration aliases.

The ***configuration information*** section of the *EXTENSIBLE EDITOR* file includes the following fields:

CONFIGURATION (.01): This is the configuration name and can be up to 8 characters in length. The file is delivered populated with four configurations: DEFAULT, HELP, BROWSE, and PROG. None of these should be deleted, but with the exception of the .01 field, they may be modified to meet institutional needs. The DEFAULT configuration is, as the name implies, the default configuration used by the editor when none is specified. The HELP configuration is used to invoke the hypertext browser. The BROWSE configuration is provided as a read-only text browser. It may be used by future applications that will require this capability, such as narrative report browsers. The PROG configuration is optimized for editing M language routines and may be invoked through the REDIT routine (see section 4.4).

VERSION (.5): This is a version identifier for the configuration. This is not necessarily the same as the version number of the software. When a configuration is modified significantly, the version identifier can be updated to reflect the change. The version identifier is loaded into the local variable RGVER and can be displayed in message text by including it as a replaceable parameter.

CREATION DATE (.7): This is the date on which the configuration was created. It is a non-editable field that is triggered by modification to the .01 field. It is used during the installation process to determine which configurations have been newly created and require pointer resolution.

1. Related Files

HELP FRAME (GENERAL) (4): This free text field names the entry level help frame for this configuration. This provides access to general help for the configuration. If null, no general help will be provided. The text must match the name (or internal entry number) of an existing help frame within the *HELP FRAME* file.

HELP FRAME (CONTEXT-SENSITIVE) (4.5): This free text field names a help frame that contains all keywords that may be used for context-sensitive help. The help frame need not contain any text as it is never displayed. The keyword entries are used as an index to other help frames. When the user invokes context-sensitive help, the editor uses the context-sensitive help frame to locate the help frame for the keyword currently under the edit cursor. The content of the field must match the name (or internal entry number) of an existing help frame within the *HELP FRAME* file.

INFORMATION TEXT (5): This is a free text field up to 50 characters in length that becomes the default message displayed on the editor message line. It may contain “|”-delimited replaceable parameters (see section 2.8).

RULER (6): This free text field may be up to 250 characters in length and contains the default tab settings for the editor. Use “=” characters to establish between tab spacing and “T” characters to denote tab stops. For example, to establish default tab stops at columns 3 and 6, specify “==T==T”. If this field is null, the editor sets up default tab stops every 8 columns starting in column #8. If a user has customized and saved tab settings, these override the settings in this field for that user.

POSTINITIALIZATION (7): This field may contain executable M code that is executed on invocation of the editor following initialization but before screen display. This allows for any additional initialization that the configuration may require (e.g., loading a report template into the main buffer).

MAXIMUM LINE LENGTH (8): This is the maximum allowable length for a line. It must not exceed the maximum length allowed by the operating system (typically 255), but may be less. The default is 255 bytes.

DISABLE TIMED READS? (8.5): If YES, the value of *DTIME* is ignored when performing reads on the keyboard buffer. If NO, the editor waits up to *DTIME* seconds for the user to provide requested input.

AUTOSAVE ON TIMEOUT? (8.7): If YES, the editor will save the contents of the main buffer if a read timeout occurs while awaiting a command. Otherwise, changes to the main buffer are lost when this occurs.

PARENT CONFIGURATION (11): Refers to another configuration entry from which this configuration will inherit its behavior. Key mappings, key macro definitions, and state variables are all inherited. Entries made in the current configuration for any of these inherited attributes overrides the corresponding entries in the parent configuration.

The **key mappings** section of the file contains one or more tables that map keystroke sequences to editing functions. These tables reside within the *KEY MAP TABLE* (1) subfile. A key map table

consists of a table identifier, a multiple containing keystroke sequence / function call pairs, and a default function call for undefined keystroke sequences.

KEYMAP TABLE (.01): This is the table identifier field and must be an integer in the range 1-999,999,999. The editor always begins with table #1 and this table must exist for every configuration. Other tables may be activated by specific function calls. For example, in the DEFAULT configuration, each programmable function key (PF1-4) activates its own key map table. If a key map table or any macro definition referenced in the table is modified, that configuration must be recompiled for the change to take effect (see *Compiling Key Mappings* in section 2.5).

CHARACTER SEQUENCE (1): This subfile contains two fields that map a character sequence to an editing function. These are:

CHARACTER SEQUENCE (.01): This free text field may be up to 200 characters in length and defines a unique keystroke sequence. The field should contain an M expression that evaluates to a text string containing the ASCII values of the keystroke sequence. If key macros (see below) are defined for the configuration, these may be used in place of the actual ASCII codes. For example, "\$C(27,65)" indicates the keystroke sequence of escape character followed by an "A". If this sequence is typed, the editor locates the entry and executes the associated M code. As another example, "PF1_\$C(66)" would match the keystroke sequence generated by pressing the PF1 key (which must be defined in the key macro subfile) followed by a "B".

EXECUTABLE CODE (1): This is the M code that will be executed when the associated keystroke sequence is generated.

DEFAULT EXECUTABLE CODE (2): This M code is executed if the typed keystroke sequence fails to match any defined sequences in the table. This might be used to generate an error message or to insert the typed characters into the text buffer.

The **key macro definition** section associates keystroke sequences with descriptive names. This makes the key map tables a little more readable and easier to maintain. There is a utility provided to facilitate creation of key macros that is described in section 2.3. The **KEY MACRO** subfile contains two fields that associate a keystroke sequence with a symbolic name. These are:

KEY MACRO (.01): This is the name of the macro and may be up to 8 alphanumeric characters in length. It must begin with a letter. The entry is case sensitive, so "PF1" is not the same as "pf1".

CHARACTER SEQUENCE (1): This free text field must contain an M expression that evaluates to a string containing the ASCII characters that comprise a keystroke sequence. You may not reference other macro names in this expression. For example, "\$C(27,32)" defines the sequence that might be generated by typing the escape key followed by the space bar.

The **state variable definition** section in the **STATE TABLE (2)** subfile defines variables that control certain editor behaviors like word wrap and screen width. The fields in this subfile are:

1. Related Files

STATE TABLE (.01): A numeric identifier for the state variable. The editor reserves values 1-999 inclusive for its own internal use. Developers creating their own extensions should prefix their identifiers with their local institution code (e.g., "583022") to prevent future conflict.

NUMBER OF POSSIBLE STATES (1): An integer in the range of 1-999 that defines the number of states possible for the variable. The editor enforces these boundaries.

LOAD USER DEFAULTS (1.5): If set to YES, and a user has private settings saved for this configuration, the setting for this state variable will be loaded instead of the default value. If NO, the default value is always loaded.

DEFAULT STATE (2): This value will be the initial state of the state variable and should not exceed the limits set by the *NUMBER OF POSSIBLE STATES* field. The lowest value for a state variable is always 0. Thus, a variable that may have 8 states can have any integer value in the inclusive range of 0-7.

EXECUTE CODE (2.3): This optional M code is executed whenever the state variable is referenced through a call to the SHST~RGEDS subroutine (see section 3.5).

STATE LABELS (3): This optional free text field, up to 240 characters in length, defines the labels that are displayed on the editor status line. Labels are separated by semicolons with the first label corresponding to state 0, the second to state 1, and so on. This field is only required for state variables whose status is to be indicated on the status line. When used, the labels should be kept short (typically less than 6 characters) and be descriptive of the current state. For each state variable that has defined labels, the editor reserves a space on the status line sufficient to display the longest label. Since this space is at a premium, long labels are not desirable. For example, word wrap is defined as state variable #10 and has the label definition ";WRAP". When in state 0 (word wrap disabled), no label is displayed. When in state 1 (word wrap enabled), "WRAP" appears on the status line.

DESCRIPTION (4): This word processing field permits annotation of the state variable and should describe what function the variable performs.

The *import filters* section defines the rules for importing other entries in the file being edited. The import function is a very useful tool, but can present a data security risk if not properly restricted. The editor assumes that other entries in an edited file are inaccessible to the user unless an import filter has been defined for that file. The import filter controls which entries a user sees and, therefore, can import. The *IMPORT FILTERS* (9) subfile contains two fields that associate a global root with an M expression that controls access to entries within the global. These are:

GLOBAL ROOT (.01): This is the global root for the file. Use the "~" character where a "^" is required. For example, the global root for controlling imports from the *MAIL MESSAGE* file is "~XMB(3.9,". When a user selects the import function, the editor locates this global root for the file being edited. If none exists, the user cannot access other entries in the file. If found, the editor invokes a lookup utility using the associated filter and displays any entries that escape the filter.

FILTER (1): This is executable M code that determines whether or not an entry can be accessed by a user. If the expression evaluates to 0, the entry is not accessible and is filtered. If nonzero,

the entry is accessible and will be displayed by the lookup routine. The expression may reference the special variable %S which is the internal identifier of the entry being evaluated. For example, to restrict access to MailMan message to only those who are recipients, use the expression: S X=\$D(^XMB(3.7,"M",%S,DUZ)). Note that the code must be of the general form "S X=<expression>".

The ***configuration aliases*** section in the *ALIASES* (10) subfile permits specification of other configurations that may serve as aliases for the current (primary) one. Once a user selects an alias, that configuration is invoked whenever the primary configuration is requested. In this manner, users may be provided a choice of alternate configurations that, for example, provide different keystroke mappings that correspond to those of commercial software with which they are more familiar.

ALIASES (.01): This multiple is a pointer to other configuration entries that may serve as aliases for the current one. This is used as a screen when presenting alias options to the user. Only those configurations that are specified here are presented to the user.

1.2 Extensible Editor Message File

This file is now obsolete and is automatically deleted upon installation. Beginning with version 2.3, all messages were stored in the *TEXT* multiple (field #4) of the VA FileMan *DIALOG* file (#.84). Any local additions to this file must be recreated in the *DIALOG* file. Only the first entry in the *TEXT* field multiple is displayed. Subsequent entries are ignored. Be sure to observe the numbering conventions recommended by **VISTA** when creating entries.

As with previous versions, replaceable parameters may be imbedded in the message by delimiting with “|” characters (see section 2.8). Inclusion of the “^” character in the message text is permitted. When using replaceable parameters in this way, be sure to set the *INTERNAL PARAMETERS NEEDED* field (#3) of the *DIALOG* file to null to prevent VA FileMan from attempting to interpret them.

It should be noted that this file is used by all configurations. Therefore, any change to the file will be reflected in all configurations displaying the modified message.

1.3 Extensible Editor Spellchecker File

For general word processing, this file provides the ability to spell check a document. The inclusion of a Soundex (AC) cross-reference permits the presentation of suggested spellings for misspelled words. The file contains a single field.

WORD (.01): A free text field of up to 100 characters.

2. Creating Configurations

2.1 Introduction

The Extensible Editor, by design, is highly configurable. Because of this, the package manager has a great deal of flexibility and control over the behavior of the editor. An attempt has been made to balance flexibility with ease of modification. However, many modifications will nonetheless require a knowledge of the M programming language and the editor's API (which is discussed in detail in the next section). Before modifying one of the distributed configurations, we recommend making a copy of the configuration under another name using VA FileMan's *Transfer File Entries* option. Then, if the configuration becomes hopelessly corrupted, it may be deleted and then restored from its copy using the same option.

We also recommend using one of the distributed configurations as a starting point for creating new configurations. This can be accomplished in one of two ways. Again using the *Transfer File Entries* option, simply select the existing configuration that most closely approximates what you need, transfer it to your new configuration (be careful not to delete the original!), and then edit the new configuration to meet your needs. Alternatively, you may create a new configuration entry using VA FileMan and specify a parent configuration to inherit from. Then you need add only those additional features (or override existing features) that are specific to the new configuration. Note that changes to the parent configuration would be reflected in your new configuration (unless the latter has overridden them with its own definitions).

2.2 Distributed Configurations

The *EXTENSIBLE EDITOR* file is delivered with four predefined configurations. With the exception of the *CONFIGURATION* (.01) field, these may be modified to meet local needs, but care should be exercised in doing so. The supplied configurations are:

DEFAULT: This is the default configuration used by the editor when none is specified. It is designed for general text editing and is appropriate for most applications. It is this configuration that will typically be used when invoked as an alternate editor under **VISTA**. There is no context-sensitive help provided with this configuration. There is general help which describes the operation of the editor in detail. Because the help frames distributed with the editor describe its behavior in the unmodified state, you may need to edit help frames to reflect your local modifications.

HELP: This configuration is used by the hypertext browser for displaying help frames. The Extensible Editor implements hypertext help using the **VISTA HELP FRAME** file. The **VISTA** Kernel provides utilities for maintaining this file. The **HELP** configuration greatly improves on the **VISTA** hypertext help model while using precisely the same files provided by the Kernel. In fact, applications may display their help frames by invoking the editor with this configuration through a standard API call (see section 3.5). The **HELP** configuration has far fewer key map entries than the **DEFAULT** configuration because it requires only a few commands for navigating hypertext links. It also requires fewer state variables and does not define any import filters (because it is read-only).

2. Creating Configurations

BROWSE: This configuration is very similar to the *HELP* configuration. Its key map tables contain mostly cursor positioning commands and no text editing functions. It is designed to be a read-only browser of text. It will be used by future applications that require a text browser.

PROG: This configuration is a derivative of the *DEFAULT* configuration. It is tailored to the specialized needs of M programmers. It defines extensions to the editor that are useful to the programmer. In addition, it provides general and context-sensitive help that will assist in developing software. These help frames are not fully populated in this release. In future releases, we hope to provide a full treatise of the M programming language and descriptions of all *VISTA* API's.

2.3 Creating Keystroke Macros

Keystroke macros improve the readability and maintainability of the key map tables. Referencing PGUP instead of \$C(27,91,53,126) is certainly preferable. In addition, if you are using terminals that generate a different character sequence for this key, it is easier to modify the associated macro than try to find where this sequence occurs in the key map table.

Keystroke macros can be created by modifying the configuration using VA FileMan. However, this requires you to know in advance what the keystroke sequences will be. To simplify this task, we provide the RGEDKD utility. This may be invoked from the command line as D ^RGEDKD or can be defined in the option file and invoked from a *VISTA* menu. The utility will ask which configuration you are modifying. Select the configuration for which you will be defining key macros and you will be prompted for a macro name followed by a prompt to type the key to be associated with the macro. The utility stores the ASCII sequence generated by the typed key in the appropriate format within the key macro definition table. Multiple macros may be quickly defined in this manner.

2.4 Establishing Key Mappings

You create and modify key mappings using the VA FileMan *Enter/Edit* option on the *EXTENSIBLE EDITOR* file. First select the configuration to be modified and then the identifier of the key map table you wish to change. The editor always starts with the table numbered "1". Additional tables are accessed through specific API calls. Each table has two fields, a multiple called *CHARACTER SEQUENCE* that defines keystroke sequence / editor function mappings and one called *DEFAULT EXECUTABLE CODE* which is invoked whenever a keystroke sequence fails to match one listed in the table. The *CHARACTER SEQUENCE* multiple has two fields. The *CHARACTER SEQUENCE* field defines the keystroke sequence that when generated invokes the executable M code found in the second field called *EXECUTABLE CODE*.

If all this sounds complicated, it really is not, though defining large key map tables can be a lengthy task. Consider the following sample configuration (field names are *italicized*, field values are **bolded**):

CONFIGURATION: **SAMPLE**

```

KEY MAP TABLE: 1
DEFAULT EXECUTABLE CODE: D INSERT^RGED0(RGCH1)
CHARACTER SEQUENCE: $C(5)
EXECUTABLE CODE: D END^RGED2
CHARACTER SEQUENCE: PF1
EXECUTABLE CODE: D CMD^RGED0(2)
KEY MAP TABLE: 2
DEFAULT EXECUTABLE CODE: W $$PRMPT^RGED2("Bad command",0)
CHARACTER SEQUENCE: H
EXECUTABLE CODE: D GEN^RGEDH

```

This excerpt from a hypothetical configuration called SAMPLE defines two key map tables numbered 1 and 2. Table 1 defines two character sequences, one for CONTROL-E (ASCII character #5) and one for the PF1 key (a macro defined elsewhere in this configuration). If CONTROL-E is typed, the editor finds this in the table and executes the associated code (D END^RGED2) which moves the cursor to the end of the current line. If the keystroke sequence corresponding to that defined in the PF1 macro is generated, the editor executes D CMD^RGED0(2) which activates key map table 2 (the argument of the subroutine). The editor then refers to table 2 for the next keystroke. If the "H" key is typed, the editor executes D GEN^RGEDH which invokes the editor help. Any other key is trapped by the default executable code causing the message "Bad command" to be displayed on the message line. Likewise, in table 1 if something other than PF1 or CONTROL-E is typed, the code D INSERT^RGED0(RGCH1) is executed causing insertion of the text corresponding to the keystroke sequence at the current edit position (in accordance with the philosophy: if it's not a command, it's text).

There is one special case worthy of mention. On some systems (e.g., IBM-compatible PC's), the ASCII null character (#0) may be generated as part of a key sequence. Since M does not allow the ASCII null character in strings or subscripts, this character is converted to the ASCII character #255 when received from the keyboard buffer. Thus, when defining key mappings, you must make this substitution or an error will occur upon compilation. The key macro utility (RGEDKD) automatically performs this conversion for you.

2.5 Compiling Key Mappings

Any changes to the key mappings (directly, or indirectly by modifying a macro definition referenced by a key mapping) necessitate compilation of the configuration to take effect. For efficiency sake, the editor does not access the key map tables directly. Rather, it accesses a special global created during the compilation process to locate keystroke sequences and executable code. This global, called RGED, contains not only compiled key mappings, but also user-defined default editor settings and command macros. It is not VA FileMan-compatible and should not be directly modified.

To compile key mappings, enter D ^RGEDCMP from the command line or define this as an option in the *OPTION* file and invoke it from your **VISTA** menu. This utility will prompt you for a configuration name and will proceed to compile the configuration you specify. Entering ALL at this prompt will cause all configurations to be recompiled. This procedure is automatically done during installation of the editor to create the ^RGED global.

Compiler errors may be of two types. The first occurs when the *CHARACTER SEQUENCE* field contains an invalid expression or references a nonexistent keystroke macro. When this occurs, the compiler ignores the entry and continues. The second type of error occurs when a duplicate keystroke mapping is detected within a key map table. When this occurs, the compiler ignores the duplicate definition and continues. In either case, the compiler signals that the error has occurred and displays information helpful in locating the problem entry.

2.6 The State Variables

The editor makes use of several variables that control many facets of its behavior. These variables are loaded from the *STATE TABLE* subfile of the *EXTENSIBLE EDITOR* file or from user-defined defaults into a local array called RGST. Each state variable is identified by an integer index with values below 1,000 reserved for use by the editor. Developers may define their own entries by prefixing a three digit index with their site code (e.g., "583235", where 583 is the site code and 235 is the variable identifier). This will prevent future conflicts in numbering conventions.

For new configurations, you should provide entries for the 15 predefined state variables. For information on predefined state variables, see section 3.3.

2.7 Creating Import Filters

The Extensible Editor provides a powerful text import feature that allows the user to import text from a variety of sources. However, providing unlimited access to external text introduces a potential data security concern in any multi-user environment. To address this concern, the editor assumes users without programmer privileges have no access rights to other entries in the file being edited unless explicitly granted by the configuration. These access restrictions are enforced through the use of import filters.

The package manager may create import filters for a configuration by editing the *IMPORT FILTER* subfile of the *EXTENSIBLE EDITOR* file. This subfile has two fields, the *GLOBAL ROOT* field specifies which global entries are being regulated and the *FILTER* field specifies the criteria governing access. An entry must exist for every file that is to be accessed. Otherwise, no access will be granted.

To understand how to specify global roots, the package manager must have an understanding of the global structure of the file of interest. When an attempt is made to import another file entry, the editor must locate the file's index for entries at that level. Since the editor has no linkage to the data dictionary of the file being edited (only the global root passed as the first argument to the editor is known), it must examine the global directly to locate the index. This is accomplished by starting with the global root passed to the editor as a formal parameter (i.e., the RGDIC variable) and testing for the presence of a "B" cross-reference at the penultimate subscript level. If one is not found, the editor aborts the import operation. Otherwise, the editor forms a global root for the index containing subscripts up to but not including the level of the cross-reference. The editor then examines the import filters defined for the active configuration to see if any match the new global

root. If no match is found, access is prohibited. If a match is found, the editor performs a file lookup using the associated filter.

To see how this works, let us examine the import filter that is distributed with the DEFAULT configuration that permits access to other entries in the *MAIL MESSAGE* file when the editor is invoked by MailMan. This filter limits mail message access to its recipients. MailMan stores mail messages in this file in a word processing field. It is this field that is being edited when you compose a mail message. If you were to examine the value of RGDIC while editing a mail message, it might look something like this:

```
^XMB(3.9,73,2)
```

The first subscript is the VA FileMan file number. The second subscript corresponds to the internal message identifier and its value will vary. The third subscript is always 2 and is the internal identifier for the word processing field. To locate the global's index, the editor first tests for the presence of a "B" index at the n-1 subscript level, i.e., ^XMB(3.9,"B"). It then strips off all but n-2 subscripts to form the global root "^XMB(3.9)" which it locates in the *IMPORT FILTER* subfile as entry "~XMB(3.9)" (remember, the "~" character is used in this field wherever a "^" is required). It then invokes a file lookup by the located index using the associated filter:

```
S X=$D(^XMB(3.7,"M",%S,DUZ))
```

This filter screens each mail message entry whose subject partially matches the user's input to determine if the current user is a recipient. The variable %S always represents the internal identifier of the entry being examined (in this case, the second subscript of the original global root). If the user is a recipient of the message being examined, this node will exist, the expression evaluates to a nonzero value, and the selection will be displayed. If the expression evaluates to zero, the user will never see that entry (i.e., it is filtered).

If you wish to permit unlimited access to a particular file, associate its global root with the filter "S X=1". Again, remember that import filters apply only to the importing of file entries and are ignored if the user has programmer access privileges.

2.8 Using Replaceable Parameters

An editor message or prompt may contain replaceable parameters that are evaluated prior to displaying the message. These are designated by enclosing them within paired "|" delimiters. If the replaceable parameter requires the use of the "^" character, you may use the "~" character in its place. This substitution is required if the message is to be stored in the *DIALOG* file, since VA FileMan will not accept the "^" character.

For example, the message text "Editor Version |RGVER|" would display as "Editor Version 2.6", if the value of the variable RGVER was 2.6. Replaceable parameters can be any valid M expression. To imbed the "|" character as a literal, use "||". Care should be taken when extrinsic functions are used, since they may have undesirable side effects.

2. Creating Configurations

Because VA FileMan also allows for the inclusion of special “|”-delimited variables in the *DIALOG* file, be sure to set the *INTERNAL PARAMETERS NEEDED* field to null if you want the editor, and not VA FileMan, to interpret the imbedded parameters.

2.9 Other Configuration Parameters

The remaining configuration parameters are detailed in chapter 1 and are fairly self-explanatory. Fill these in according to the specific needs of your configuration.

3. Application Programming Interface

3.1 Introduction

The Extensible Editor defines an extensive set of subroutines that may be referenced in key mappings or by extensions to the editor. In addition, a number of application variables exist that define the editing environment. When writing extensions for the editor, pay close attention to the rules governing appropriate use of these variables. Never modify an application variable unless it is explicitly allowed. Doing so may introduce erratic behavior and undesirable results.

Unlike most editors, the Extensible Editor was designed to be an open environment. Using its well-defined interface, developers may create M programs that significantly extend the capabilities of the editor beyond those provided in the current release. Indeed, such development is strongly encouraged. Those individuals who create universally useful extensions are urged to submit these to our department for inclusion in future releases. Full acknowledgment will be given to those individuals who so contribute.

3.2 Application Variables

The following variables are defined by the editor. All fall under the RG namespace which is the reserved namespace for this development site. Unless explicitly defined to be modifiable, they should be treated as read-only. For those that may be modified directly, close attention should be paid to any constraints that may accompany their modification. Failure to do so may cause erratic or otherwise undesirable behavior.

RGBEL	Contains the BELL character or is null if the bell has been disabled.
RGBMC	Column position of the current bookmark, or 0 if none defined.
RGBMR	Row position of the current bookmark, or 0 if none defined.
RGBUF	Internal identifier of the buffer currently active. Identifiers are of the format "RGEDx" where x is null if it is the main buffer or the letter or number of the active buffer otherwise.
RGCF	The internal identifier of the active configuration. This is actually the record number within the <i>EXTENSIBLE EDITOR</i> file.

3. Application Programming Interface

RGCF2	If a configuration has been aliased as another, this contains the internal identifier of the original configuration, otherwise it will be 0.
RGCH	The string of unprocessed keystrokes.
RGCH1	Last character received from keyboard buffer.
RGCHG	Set to a nonzero value whenever the current buffer is modified. This indicates to the editor that a modification has taken place. You must set this variable to 1 if you modify the active buffer outside a standard API subroutine call. Failure to do so or setting it to 0 may result in changes being lost when the editor exits.
RGCLR	Contains the VT100 command sequence to clear the screen.
RGCOL	Column position of the current edit position. The first column is numbered 1.
RGCTL1	A string containing non-displayable ASCII characters.
RGCTL2	A string containing the display equivalents of the characters in RGCTL1.
RGDIC	This is the global root of the word-processing field being edited. This is one of the formal parameters passed on invocation of the editor.
RGEOL	Contains the VT100 command sequence to clear to the end of the line.
RGEOS	Contains the VT100 command sequence to clear to the end of the screen.
RGESC	Contains the escape character, ASCII 27.
RGFIN	If this variable exists, the editor will exit upon return from the subroutine that created it. You may set this variable, but do so with care. The editor does not save any changes, but exits immediately upon return.
RGHION	The VT100 escape sequence for activating the highlight attribute.
RGHLP	This contains the text of the default information message as defined in the configuration's <i>INFORMATION TEXT</i> field. This is the message displayed on the editor's message line when no other message is active. You may modify this if you wish, but its length should not exceed 50 bytes. You may include replaceable parameters.
RGITER	This is the value of the current iteration count. If your routine supports multiple iteration, it may use this variable and decrement it with each iteration. The iteration count is reset to 0 following execution of the first command after it is set.

RGITER2	This is the pending iteration count. The ITER^RGED1 subroutine sets this variable, which becomes the active iteration count (RGITER) on the next command fetch cycle.
RGLC	This is the line count for the active buffer.
RGLFT	This is the character offset of the leftmost character displayed. Because the editor supports horizontal text scrolling, the leftmost character may not be the first character in a line.
RGLN	This is the text of the line currently being edited. You may modify this as needed. The editor automatically writes changes back to the buffer whenever the current line changes or when the UPDATE^RGED2 subroutine is called.
RGMAX	This is the maximum allowable line length. It is set according to the value specified in the configuration's <i>MAXIMUM LINE LENGTH</i> field.
RGMSG	If nonzero, the editor will redisplay the default information message following execution of the next command. Normally, you should not need to modify this variable as the editor PRMPT^RGED2 subroutine sets it when a message is displayed.
RGNORM	The VT100 escape sequence that turns off all special display attributes.
RGOBJ	The name of the object being edited. This is displayed in the top right-hand corner of the screen. It may contain replaceable parameters. It may be modified if desired. This is one of the formal parameters passed on invocation of the editor.
RGPID	This is the instance identifier of the current invocation of the editor. It is one of the formal parameters optionally passed to the editor. On first invocation, it will normally equate to the value of the \$JOB system variable + .0001.. If the editor is entered recursively, this parameter is automatically incremented by .0001 to maintain a unique context.
RGROW	The current row being edited. The first row is designated row 1.
RGRULER	The ruler line. Tab stops are marked by "T". Intervening space is denoted by "=".
RGRVON	The VT100 character sequence for activating reverse video.
RGSC1	The column position of the first anchor point for a text selection.
RGSC2	The column position of the second anchor point for a text selection.

3. Application Programming Interface

RGSR1	The row position of the first anchor point for a text selection. Will always be zero if no selection is active.
RGSR2	The row position of the second anchor point for a text selection.
RGSRCH	This is the value of the last search string used.
RGST	This is the table of state variables. See section 3.3 for more details.
RGTBL	This is the identifier of the active key map table. It may be modified with caution. Setting it to a nonexistent identifier will cause the editor not to respond to any keystrokes.
RGTOP	This is the row number of the first text line displayed on the screen.
RGTTL	This is the text title displayed in the upper left-hand corner of the screen. It is one of the formal parameters passed to the editor on invocation. It can be modified if desired. It may contain replaceable parameters.
RGUNON	The VT100 escape sequence for activating the underline attribute.
RGVER	The version identifier for this configuration. This is the value of the <i>VERSION</i> field.
RGX	The X-coordinate value of the current edit position.
RGX1	The X-coordinate value of the leftmost column of the text window. Should always be 1.
RGX2	The X-coordinate value of the last column of the text window. Should be 80 or 132 depending upon the current width setting.
RGY	The Y-coordinate value of the current edit position.
RGY1	The Y-coordinate value of the first row of the text window.
RGY2	The Y-coordinate value of the last row of the text window.

3.3 The State Variable Array

State variables control several editor characteristics. They are loaded from the *STATE TABLE* subfile of the *EXTENSIBLE EDITOR* file or from user-defined defaults into a local array called *RGST* during editor initialization.

The format for the RGST variable follows (n represents the variable identifier). The developer should make no direct modifications to this array.

RGST(n)	Contains the current value for the state variable.
RGST(n,-1)	Contains the number of possible states.
RGST(n,-2)	Contains the ordinal position on the status line. Not defined if there are no state labels associated with the variable.
RGST(n,-3)	Contains any executable M code associated with the variable. Not defined if no such association exists.
RGST(n,-4)	Nonzero if this is a savable parameter. Controls whether or not a user can save the variable's current value as a private default.
RGST(n,m)	Where $m \geq 0$, contains the state label associated with the state value m.

The following state variables are defined by the Extensible Editor:

1	This is a value that identifies the currently active buffer. Valid values are 0 for the main buffer, 1 for the delete buffer (which should never occur, since this buffer should never be active), 2-27 for buffers A through Z, and 28-37 for buffers 0 through 9. Do not modify this variable using SHST^RGEDS. It is automatically modified by SWITCH^RGED5.
2	If 0, the editor is in insert mode. All typed text is inserted at the current edit position, shifting characters to the right as necessary. If 1, the editor is in overwrite (replace) mode. All typed text overwrites whatever exists at the current edit position. You may use SHST^RGEDS to modify this setting.
3	If 0, the editor performs text searches from the current position toward the end of the buffer. If 1, text searches progress from the current position toward the beginning of the buffer. You may modify this setting using SHST^RGEDS.
4	If 0, text searches are case insensitive. If 1, searches are case sensitive. Use SHST^RGEDS to modify this setting.
5	If 1, edit position tracking is enabled. The editor displays the current edit position and the ASCII value of the character at that position on the right-hand side of the message line. If 0, cursor position tracking is disabled. You may use SHST^RGEDS to change this setting.

3. Application Programming Interface

6	If 0, the editor displays the ruler line. If 1, the editor displays the ruler line with the column position indicator active. If 2, the editor suppresses display of the ruler line. You may modify this setting with SHST^RGEDS.
7	This is the left margin setting. Set this through a call to LM^RGED1 only.
8	This is the right margin setting. Set this through a call to RM^RGED1 only.
9	If 0, the bell tone is active. If 1, the bell tone is silenced. The editor uses the bell tone to signal an error. You may modify this setting with SHST^RGEDS.
10	If 0, word wrap is disabled. If 1, word wrap is enabled. Use SHST^RGEDS to modify this setting.
11	This is the maximum display width. It should be either 80 or 132, depending on the display mode. Modify this only through a call to SCRN^RGEDS.
12	This is the maximum display height in rows. It is typically 24 or 25. It may be less than the actual number of rows supported by the display, but should never be more. Modify this only through a call to SCRN^RGEDS.
13	If 1, the editor supports imbedded hypertext. This setting is normally used only for the HELP configuration. If 0, the editor does not recognize imbedded hypertext and displays non-printable characters (ASCII codes 0 through 26, 28 through 31, and 127 through 255) as the tilde character or the dollar sign in the case of the escape character (ASCII code 27).. This is the normal setting for an editing session. If 2, the editor does not recognize imbedded hypertext and displays all text in raw form. This setting may be used in text browsing applications that contain imbedded formatting information (e.g., VT100 escape sequences). Modifying this setting at run time will produce unpredictable results.
14	If 0, text selection is not active. If 1, text selection is currently active. Use SELECT^RGED5 to modify this setting.
15	If 1, the text editor performs text insertion on the display by refreshing to the end of the line. This is necessary for VT100 terminal emulations that do not support the text insertion escape command. If 0, the editor uses the more efficient text insertion escape command. You may use SHST^RGEDS to change this setting.
16	If 0, the tab function causes the edit cursor to be positioned at the next tab stop leaving any intervening text unaltered (soft tab mode). If 1, the text to the right of the edit cursor is shifted to the right to align with the next tab stop before repositioning the cursor at the tab stop (hard tab mode).

3.4 Text Buffers

The Extensible Editor stores edited text in one of 38 text buffers: the main buffer, the delete buffer, 26 temporary buffers, and 10 persistent buffers. These buffers are stored within the ^TMP global and may be accessed and modified directly if desired. The format is:

^TMP(RGPID,RGBUF,<line #>)

where RGPID is the instance identifier of the editor (see previous section) and RGBUF is the active buffer. If you wish to access a buffer other than the active one, you may supply the actual buffer identifier rather than use the application variable RGBUF. The format for buffer identifiers is "RGEDx" where x is null for the main buffer, "@" for the delete buffer, "A"-"Z" for the respective temporary buffer, or "0"-"9" for the respective persistent buffer. The <line #> subscript is simply the ordinal number of the line within the buffer with the first line numbering 1. Thus, to reference the 6th line in the "A" ancillary buffer you would use the global reference ^TMP(RGPID,"RGEDA",6).

Some caveats surround the use of buffers. First, if the buffer you are accessing is not the active one, the application variables defining edit position and other buffer characteristics (e.g., RGROW and RGLN) will not apply to that buffer. If this information is needed, you should first make that buffer active through a call to the SWITCH^RGED5 subroutine. This subroutine updates all relevant application variables to reflect the status of the activated buffer. Second, you should always make a call to UPDATE^RGED2 before directly accessing the active buffer. The Extensible Editor buffers changes made to the current line in the variable RGLN. These changes are automatically written back to the buffer when the edit position moves to a different line or by API subroutines that directly access the active buffer. To insure that the active buffer reflects all edits, you must explicitly update prior to accessing it.

If you directly add or remove lines from a buffer you should observe the following constraints. First, this should only be done on the active buffer. If the buffer you wish to so modify is not active, make it so by a call to SWITCH^RGED5. Second, call UPDATE^RGED2 before modifying the buffer for the reasons given above. Third, be certain to update the application variable RGLC to reflect the new line count. Fourth, following the modification, set RGLN equal to the contents of the line at the current edit position. Fifth, if following a line deletion, the current edit position points beyond the end of the buffer, you should make a call to MOVETO^RGED2 to restore it to a valid position within the buffer. And finally, to update the screen to reflect the changes, call PAINT^RGEDS.

In addition to the caveats listed above, persistent buffers have special considerations. The contents of persistent buffers are stored with the user's configuration information. Thus, they persist across editing sessions and are shared across all configurations for a given user. However, for efficiency persistent buffers are not loaded until they are first referenced and are not saved until the editor exits. All API calls that manipulate buffers automatically load a persistent buffer when it is first referenced. If you wish to directly manipulate the contents of one of these buffers and want to insure that it is loaded, a call to BGETN^RGED5 will do this for you.

3.5 Subroutine Library

The Extensible Editor provides a rich library of subroutines that define a complete set of basic editing and display functions that may be combined in new ways to create useful extensions. Though a number of entry points exist within the editor that are not documented here, it is strongly recommended that only documented entry points be utilized. With the exception of ^RGED, these subroutines may only be invoked from within the editor environment.

^RGED(RGDIC, RGTTL, RGOBJ, RGCF, RGPID)

This is the entry point for invoking the Extensible Editor. It may be invoked outside the editor environment or recursively from within the editor environment. Only the first argument is required. The arguments are:

RGDIC	This is the global root of the word-processing field being edited. The contents of the word-processing field are loaded into the main buffer during editor initialization.
RGTTL (optional)	The title of the editing session, displayed in the upper left-hand corner of the screen. Replaceable parameters are allowed.
RGOBJ (optional)	The name of the object being edited, displayed in the upper right-hand corner of the screen. Replaceable parameters are allowed.
RGCF (optional)	The name of the configuration that is to be invoked. This is the name of the .01 field in the <i>Extensible Editor</i> file. If not specified, the editor uses the DEFAULT configuration.
RGPID (optional)	This is the instance identifier for the editor. Normally, this parameter is not passed and defaults to the value of \$JOB. If the editor is invoked recursively (i.e., from within the editor itself), this parameter should be the current value of RGPID. The editor adds .0001 to this value to maintain a unique context across all instances.

Example: D ^RGED(DIC)

Loads the word-processing field referenced by the local variable DIC into the main buffer and invokes the DEFAULT configuration.

Example: D ^RGED("^XMB(3.9,1428,2,","MailMan","Re: meeting","MAIL")

Loads the text of MailMan message #1428 (if any exists) into the main buffer and invokes a locally created configuration called MAIL. "MailMan" is displayed in the upper left and "Re: meeting" in the upper right of the screen. Note that if the word processing field referenced by the first argument does not exist, the buffer will initially be empty and the word-processing field will be created on exit from the editor (if changes are saved).

`$$^RGXY(RGX, RGY)`

Position the cursor to the display coordinates given by RGX and RGY. The upper left display position is (0,0). \$X and \$Y are set to the new coordinates. This is implemented as an extrinsic function and always returns null.

RGX (optional)	Specifies the X-coordinate for the cursor. If not specified, defaults to 0.
RGY (optional)	Specifies the Y-coordinate for the cursor. If not specified, defaults to 0.

Example: `W $$^RGXY(10,5),"TEXT"`

Positions the display cursor to coordinates (10,5) and displays "TEXT".

Example: `S X=$$^RGXY()`

Moves the display cursor to the home position. X will be null upon completion.

ALIAS^RGED4(RGCF)

Establish the configuration named in RGCF as a private alias for the user's current configuration. When the editor is subsequently requested to load the original configuration for this user, the alias is loaded instead. This allows privileged users to create their own private configurations to be used in place of the public ones. Note that aliasing is non-recursive. That is, you may not alias an aliased configuration. Because aliases are private to the user, the user must be identified to establish an alias (i.e., the DUZ must be defined).

RGCF (optional)	The name of the configuration to be established as an alias for the current one. If not specified, the editor prompts for a configuration.
--------------------	--

Example: D ALIAS^RGED4("DKM")

A configuration called "DKM" becomes an alias for the current configuration. If the current configuration is "DEFAULT" when this call is performed, the editor will load the "DKM" configuration for this user the next time the "DEFAULT" configuration is requested.

ASCII^RGEDX2

Display a table of ASCII characters.

This is an exported extension for the PROG configuration of the editor.

BACK^RGEDH

Return to the previous help frame. Valid only when a help-style configuration is active.

BDEL^RGED5(RGBUF)

Delete the buffer named in RGBUF. If this is the active buffer, the editor switches to the main buffer following the deletion. You may not delete the main buffer. Once deleted, a buffer's contents are irretrievable.

RGBUF (optional)	The name of the buffer that is to be deleted. Must be a single character in the range "A"-"Z", "0"-"9", or "@". If not specified, the editor prompts for a buffer name.
---------------------	---

Example: D BDEL^RGED5("M")

Deletes the contents of the M buffer. If this is also the active buffer, the editor switches to the main buffer following the deletion.

\$\$BGETN^RGED5(RGBUF, RG Prompt)

Return the fully formed subscript for the specified buffer. If the specified buffer is a persistent buffer and has not yet been loaded, the stored contents of the buffer is loaded into local storage. This function is called indirectly by all buffer manipulation API's. It is generally not required except in situations where a buffer's contents is to be directly manipulated.

RGBUF (optional)	The name of the buffer that is to be deleted. Must be a single character in the range "A"-"Z", "0"-"9", or "@". If not specified, the editor prompts for a buffer name using RG Prompt
RG Prompt (optional)	The prompt to be used when the first parameter is not specified.
Return value	The fully formed subscript used to reference the buffer contents in the ^TMP global. If the buffer specified was not valid, the return value is null.

Example: S X=\$\$BGETN^RGED5(3)

Returns the value "RGED3" and loads the persistent buffer if necessary.

BELL^RGED0

Set the value of RGBEL according to the setting of the bell activation state variable (#9). If the bell is active, RGBEL contains ASCII character code 7. Otherwise it is null.

Normally this subroutine is called only by the SHST^RGEDS subroutine when modifying the state of bell activation to automatically update the value of the RGBEL variable. It may be called in special settings where the value of RGBEL may require updating.

BKTB^RGED2(RGITER)

Move left RGITER tab stops. If there are no more tab stops, no action is taken.

RGITER (optional)	The number of tab stops to traverse. If not specified, defaults to one.
----------------------	---

Example: D BKTB^RGED2(4)

Move left 4 tab stops.

BKTB^RGEDH

Move to the previous hypertext link within a help frame. Valid only when a help-style configuration is active.

BMFND^RGED1

Move the current edit position to the location of the current bookmark. Bookmarks may be set by the *BMSET^RGED1* subroutine.

BMSET^RGED1(RGR,RGC)

Set a bookmark at the specified row and column position. The *BMFND^RGED1* subroutine can then be used to move the current edit position to this location. Separate bookmarks are maintained for each buffer. If no bookmark has been defined, the editor signals an error.

RGR (optional)	The number of the line to save as a bookmark. Defaults to the current line.
RGC (optional)	The column position to be saved as a bookmark. Defaults to the current column.

Example: D BMSET^RGED1(10,5)

Sets a bookmark at line 10, column 5.

Example: D BMSET^RGED1()

Sets a bookmark at the current edit position.

BREAK^RGED1

3. Application Programming Interface

BREAK^RGED1

Insert a line break at the current edit position.

BTM^RGED2

Move the current edit position to the end of the buffer. The column remains unchanged.

CALC^RGEDX2

Display the value of an M language expression. The editor prompts for an expression and then displays its value or signals an error if it is invalid. Any M language expression can be used. Reference extrinsic functions with care as the editor has no control over what operations a function might perform.

The return value of the expression is automatically stored in the delete buffer and may be inserted in the active buffer using the undelete or paste operations.

This is an exported extension for the PROG configuration of the editor.

CLRTAB^RGED1

Clear all tab stops.

CMD^RGED0(RGTBL,RGP)

Enter command mode using the key map table specified by RGTBL. In command mode, the user is prompted for a single keystroke. The editor then executes the code associated with the generated character sequence in the selected table and returns to the previously active key map table.

RGTBL	The value of the key map table to be referenced.
RGP (optional)	The prompt to be displayed. Defaults to message #1 which displays "PFx command:" where <i>x</i> is the # of the PF key invoking the command. Special applications of this subroutine will want to use their own prompts.

Example: D CMD^RGED0(2)

Enters command mode using key map table 2, and displays the prompt "PF1 command:".

CMNT^RGEDX2

Reformat comments in an M language routine. Comments that follow commands are left justified beginning at the right margin setting. Comments on lines with no commands are left justified beginning at the left margin.

This is an exported extension for the PROG configuration of the editor.

To illustrate the effect of this operation, consider the following sample M code:

```
TEST          ;; This comment will be moved left
              D X,Y,Z      ; This comment will be moved right
                              ; This comment will also be moved left
              S X=X+1
```

After formatting, the code might look like the following (depending, of course, on the left and right margin settings):

```
TEST  ;; This comment will be moved left
      D X,Y,Z                          ; This comment will be moved right
      ; This comment will also be moved left
      S X=X+1
```

COPY^RGED5(RGBUF,RGCUT)

Copy selected text from the active buffer to the destination buffer named in RGBUF. If RGCUT is nonzero, the original text is deleted (i.e., a cut operation is performed), otherwise it remains intact (a copy operation). The destination buffer cannot be the active buffer nor can it be the main buffer. Any text present in the destination buffer prior to the operation is lost.

This editor signals an error if no text selection is active.

RGBUF (optional)	The name of the buffer that is to receive the copied text. This must be a single character in the range "A"-"Z", "0"-"9" or "@". If not specified, the editor prompts for a buffer name.
RGCUT (optional)	If specified and nonzero, the selected text is removed from the active buffer following the copy operation.

Example: D COPY^RGED5("Z")

Copies selected text from the active buffer to the Z buffer.

Example: D COPY^RGED5("@",1)

Cuts selected text from the active buffer to the delete buffer. This is identical to a block delete operation.

Example: D COPY^RGED5()

Copies selected text from the active buffer to an unspecified buffer. The editor will prompt for the name of the destination buffer.

CSH^RGEDH

Invoke context-sensitive help using the **HELP** configuration. The editor locates the keyword at the current edit position and attempts a lookup in the keyword multiple associated with the help frame specified in the *HELP FRAME (CONTEXT-SENSITIVE)* field of the *EXTENSIBLE EDITOR FILE*. If found, the help frame associated with the keyword is loaded. Otherwise, the editor signals an error.

CUT^RGED5(RGBUF)

Copy selected text from the active buffer to the destination buffer named in RGBUF. Following the copy, the text is removed from the active buffer and text selection is deactivated. The destination buffer cannot be the active buffer nor can it be the main buffer. Any text present in the destination buffer before the operation is lost.

The editor signals an error if no text selection is active.

RGBUF (optional)	The name of the buffer that is to receive the copied text. This should be a single character in the range "A"-"Z", "0"-"9" or "@". If not specified, the editor prompts for a buffer name.
---------------------	--

Example: D CUT^RGED5("Z")

Cuts the selected text from the active buffer to the Z buffer.

DELBOL^RGED4

Delete from the current edit position to the beginning of the line, shifting remaining text to the left. Deleted text is stored in the delete buffer.

DELEOL^RGED4

Delete from the current edit position to the end of the line, not including the line break. This operation effectively truncates the current line. Deleted text is stored in the delete buffer.

DELETE^RGED0(RGITER)

Delete RGITER characters from the active buffer starting at the current edit position. A line break counts as a single character and when deleted, causes the line that follows to become concatenated with the current line.

RGITER (optional)	The number of characters to be deleted. Defaults to one.
----------------------	--

Example: D DELETE^RGED0()

Deletes the single character at the current edit position.

DELLIN^RGED4(RGITER)

Delete RGITER lines starting at the current line. This operation removes entire lines, scrolling remaining text upward to fill the void. Upon completion, the delete buffer contains the last line deleted.

RGITER (optional)	The number of lines to be deleted.
----------------------	------------------------------------

Example: D DELLIN^RGED4(5)

Deletes the current line and the four lines that follow.

DOWN^RGED2(RGITER)

Move the current edit position down RGITER lines. If an attempt is made to move beyond the end of the buffer, the editor signals an error and moves to the end of the buffer.

RGITER (optional)	The number of lines to move. If not specified, defaults to 1.
----------------------	---

Example: D DOWN^RGED2()

Moves down one line.

END^RGED2

Move the current edit position to just beyond the last character in the current line.

FIND^RGEDH(RGCH,RGDIR)

Find the next hypertext link that begins with the character in RGCH. If none is found, the editor continues the search from the beginning of the help frame. If still none is found, the editor signals an error and the cursor position remains unchanged. Valid only when a help-style configuration is active.

RGCH (optional)	A single character value that is to be used to locate a hypertext link beginning with that character. If absent, the next hypertext link is located regardless of beginning character.
RGDIR (optional)	Direction of the search where 0=forward (the default) and 1=backward.

Example: D FIND^RGEDH("C")

Locate the next hypertext link beginning with the letter C.

Example: D FIND^RGEDH(,1)

Find the previous hypertext link regardless of beginning character.

FLUSH^RGEDH

FLUSH^RGEDH

Flush the type-ahead buffer.

FNEXT^RGEDF(RGITER)

Locate the RGITER occurrence of the last string searched. Equivalent to *SEARCH^RGEDF* except that the last search string is automatically used.

If no previous search string has been established, the subroutine call has no effect.

RGITER (optional)	The text string occurrence to be located. If not specified, defaults to the first.
----------------------	--

Example: D FNEXT^RGEDF()

Locates the next (or previous, depending on the setting of the search direction state variable) occurrence of the most recently searched text string.

FORMAT^RGED1(RGITER)

Format the current paragraph, performing word wrap as necessary to left justify it between the left and right margin settings. A paragraph is defined as vertically contiguous text with boundaries defined by intervening vertical white space (i.e., lines that are null or contain only spaces and/or tabs). The edit position follows the last paragraph formatted.

RGITER (optional)	The number of paragraphs to be formatted. Defaults to one.
----------------------	--

Example: D FORMAT^RGED1(3)

Formats the current paragraph and the two that follow.

GEN^RGEDH

Invoke the **HELP** editor configuration with the entry level help frame defined by the *HELP FRAME (GENERAL)* field of the *EXTENSIBLE EDITOR* file. If no such help frame is defined, the editor signals an error.

GOTO^RGED1(RGGT)

Move to the line specified by the RGGT argument.

RGGT (optional)	If numeric, specifies the absolute number of the line. If preceded by a "+" or "-", specifies the line number as an offset from the current line. If null or not specified, the editor prompts for the line number.
--------------------	---

Example: D GOTO^RGED1(10)

Moves to line # 10 in the active buffer.

Example: D GOTO^RGED1("+12")

Moves to the twelfth line following the current line.

Example: D GOTO^RGED1()

The editor prompts for the line to be located.

HELP^RGEDH(RGHFM, RGHTL, RGHKEY)

Invoke the HELP configuration using the help frame referenced in RGHFM.

RGHFM	May be either the name of the entry level help frame or the internal identifier for same.
RGHTL (optional)	The title to be given to the initial help frame. If numeric, is the internal identifier of the help frame whose name is to be used as the title. If not specified or null, the header of the entry level help frame is used. If text, directly specifies the title to be used. The title is displayed in the upper left corner of the screen.
RGHKEY (optional)	The name of the topic being displayed. This is displayed in the upper right corner of the screen.

Example: D HELP^RGEDH("HELP1","HELP","Importing")

Loads the help frame called HELP1. The title is "HELP" and the topic is "Importing".

Example: D HELP^RGEDH(123)

Loads the help frame corresponding to the internal identifier 123. The title is the value of the *HEADER* field for that entry in the *HELP FRAME* file.

Example: D HELP^RGEDH(123,"","Buffers")

Same as the preceding example, but also displays the topic "Buffers".

HLPMSG^RGEDS

Update the message line. If a message is currently displayed, it is replaced by the default information message as determined by the value of the *INFORMATION TEXT* field of the *EXTENSIBLE EDITOR* file. If cursor tracking is enabled (state variable #5), the editor also updates the cursor position information.

HOME^RGED2

Move the current edit position to the first column of the current line.

HOME^RGEDH

Return to the entry level help frame. Valid only when a help-style configuration is active.

IMPORT^RGED6(RGIMP, RGTYPE, RGITER)

Import RGITER copies of the object RGIMP of type RGTYPE into the active buffer at the current edit position. The editor defines five importable object types: VA FileMan file entry, ASCII text file, M language routine, buffer, and global reference.

VA FileMan file entries may be imported from the same file whose entry is currently being edited. The editor must be able to locate a "B" cross-reference above the level of the global root node specified in RGDIC. RGIMP will be the key value that is to be located within this cross-reference. Access to file entries is restricted through the use of import filters (see section 2.7). Users with programmer access (i.e., DUZ(0)="@"), are given unlimited access.

ASCII text files may be imported by passing the full file specification in RGIMP. The format for this will depend upon the underlying operating system which is also responsible for controlling access.

Users with programmer access may directly import M language routines by specifying the routine name in RGIMP.

Importing buffers is identical to pasting. In fact, the paste operation is implemented by a call to this subroutine. As with pasting, there is no access restriction because buffers are already private.

Users with programmer access may also directly import any word-processing field by providing the full global root specification of the field to import.

RGIMP (optional)	The object to be imported. Its format depends upon the value of RGTYPE and is detailed above. If null or not specified, the editor prompts for the value.
RGTYPE (optional)	The object type to be imported. If null or not specified, the editor prompts for the object type. This must be one of the following: B=buffer, F=VA FileMan entry, G=global reference, K=Kermit, R=M language routine, T=ASCII text file.
RGITER (optional)	The number of copies to be imported. If not specified, defaults to one.

Example: D IMPORT^RGED6("RE: MEETING", "F")

Imports the VA FileMan file entry indexed by "RE: MEETING".

Example: D IMPORT^RGED6()

Imports one copy of an unspecified object of unspecified type. The editor prompts first for object type, then object value.

Example: D IMPORT^RGED6("C:\FILE.LIS","T",2)

Imports two copies of an ASCII text file called "C:\FILE.LIS".

Example: D IMPORT^RGED6("Z","B",4)

Imports (pastes) four copies of the Z buffer.

Example: D IMPORT^RGED6("^XMB(3.9,5,2","G")

Imports the MailMan message identified by the global root "^XMB(3.9,5,2,".

Example: D IMPORT^RGED6("", "R")

Imports an unspecified M language routine. The editor prompts for the routine name.

Example: D IMPORT^RGED6(“”, ”K”)

Starts a Kermit download session.

INSERT^RGED0(RGCH, RGITER)

Insert the text in RGCH at the current edit position RGITER times. The cursor is positioned just following the inserted text. Text wraps at the right margin if line wrap is enabled.

RGCH	The text to be inserted.
RGITER (optional)	The # of times to insert the text. If not specified, this defaults to one.

Example: D INSERT^RGED0("X",5)

Inserts 5 "X"s at the current edit position.

ITER^RGED1(RGCNT)

Set the iteration counter to the designated value. This stays in effect only until the next command is executed, and then reverts back to its baseline value of 1. Note that on return from this subroutine, *RGITER2* not *RGITER* contains the new count. The editor does not assign the iteration counter variable *RGITER* the new value until it fetches the next command from the keyboard buffer. Therefore, if you are making an imbedded call to this subroutine and wish to use the iteration counter value immediately, you should reference *RGITER2* and set it to zero to avoid impacting the next typed command. If you wish to reference the iteration counter set by the preceding command cycle, reference *RGITER*.

RGCNT (optional)	The value to be assigned to the iteration counter. It must be a nonnegative integer value less than 9,999,999. 0 is allowed, but is interpreted as 1 by most commands. If not specified, the editor will prompt for a value.
---------------------	--

Example: D ITER^RGED1(),FORMAT^RGED1(RGITER2)
 S RGITER2=0

Prompts for an iteration counter, then uses the new value in a subsequent call to the paragraph formatting subroutine. The counter is then reset to zero to avoid impacting any commands that follow.

Example: D ITER^RGED1(5)

Sets the iteration counter (*RGITER2*) to 5. On the next command fetch cycle, the editor sets the variable *RGITER* to this value and clears *RGITER2*.

JOIN^RGED1

Concatenate the current line with the line that immediately follows it. This operation is invalid if the current line position is at the end of the buffer, if the current line is the last line and is not null, or if the resulting concatenation results in a line that exceeds the maximum allowable line length (as set by the *MAXIMUM LINE LENGTH* field). If any of these conditions occur, an error message is displayed and the join is aborted.

JUSTIFY^RGED4(RGJUST, RGITER)

Justify one or more lines between the left and right margins. Justification can be flush left, flush right, or center depending upon the value of RGJUST. If text selection is active at the time the subroutine is invoked, the lines contained within the text selection are justified and the RGITER argument is ignored. If text selection is not active, RGITER lines are justified starting with the current line.

RGJUST	Determines the type of justification. -1=left justification, 0=center, 1=right justification.
RGITER (optional)	The number of lines to be justified, starting with the current line. If not specified, defaults to one. If text selection is active, this argument is ignored.

Example: D JUSTIFY^RGED4(-1)

If text selection is active, left justifies all lines contained within the selection. If text selection is not active, left justifies the current line only.

Example: D JUSTIFY^RGED4(0,5)

If text selection is active, centers all lines contained within the selection, ignoring the second argument, If text selection is not active, centers the current line and the four that follow.

KEYW^RGEDH(RGKEY)

Load the help frame associated with the keyword given in RGKEY. The editor signals an error if the keyword is not found in the current help frame. Valid only when a help-style configuration is active.

RGKEY	The keyword corresponding to a hypertext link.
-------	--

Example: D KEYW^RGEDH("PASTE")

Loads the help frame associated with the keyword "PASTE".

LEFT^RGED2(RGITER)

Move the current edit position left RGITER characters. A line break counts as a single character and traversing one results in the edit position moving to the end of the previous line. If an attempt is made to move past the beginning of the buffer, the editor signals an error and moves the edit position to the first column of the first line.

RGITER (optional)	The number of character positions to move. If not specified, defaults to one.
----------------------	---

Example: D LEFT^RGED2(10)

Moves left 10 character positions.

LINK^RGEDH

Traverse the current hypertext link. The editor loads the help frame associated with the link. Valid only when a help style configuration is active.

LM^RGED1(RGC)

Set the left margin setting at the column position given by RGC. The ruler, if active, is automatically updated. The left margin cannot be within 5 columns to the left of the right margin. The editor signals an error if this occurs.

RGC (optional)	The column position that is to become the new left margin setting. If not specified, defaults to the current column position.
-------------------	---

Example: D LM^RGED1()

Sets the left margin to the current column position.

Example: D LM^RGED1(5)

Sets the left margin to column 5.

LMAR^RGED2

Move the current edit position to the left margin. The line position remains unchanged.

MDEF^RGED3(RGMACRO, RGDEF)

Load the macro definition RGDEF into the macro specified by RGMACRO. A macro definition is simply executable M code and is private to the user.

RGMACRO (optional)	The macro to define. May be a single character in the range "A"-"Z" or the "@" character. The "@" character designates a macro that is automatically executed at editor startup. If this argument is null or not specified, the editor will prompt for its value.
RGDEF (optional)	The executable M code to be associated with the macro. If not specified, the editor will prompt for its value. If null, the associated macro is deleted.

Example: D MDEF^RGED3("A","D INSERT^RGED0(\$\$^RGCVTDT(\$H))")

Defines macro A. When executed using *MEXE^RGED3*, this causes the current date and time to be inserted at the current edit position.

Example: D MDEF^RGED3("@")

Defines the startup macro. The editor will prompt the user for the M code to be associated with the macro.

Example: D MDEF^RGED3()

Defines an unspecified macro. The editor prompts for both the macro name and its associated executable code.

MEXE^RGED3(RGMACRO,RGMCNT)

Execute macro RGMACRO, RGMCNT times. Macros may be defined by *MDEF^RGED3* and are private to the user and the configuration.

RGMACRO (optional)	The name of the macro to execute. Macro names must be a single character in the range "A"-"Z" or the "@" character. If null or not specified, the editor prompts for the macro to execute.
RGMCNT (optional)	The number of times to execute the macro. If not specified, defaults to one.

Example: D MEXE^RGED3("C",RGITER)

Executes macro C, RGITER times.

Example: D MEXE^RGED3("A")

Executes macro A one time.

Example: D MEXE^RGED3()

Executes a macro one time. The editor prompts for the macro name.

MOVETO^RGED2(RGR,RGC)

Move to the edit position given by RGR and RGC. The display is updated as necessary. The editor automatically handles scrolling and text selection.

RGR	The line number to move to.
RGC	The column position to move to.

Example: D MOVETO^RGED2(4,1)

Moves to the first column of the fourth line.

MSHOW^RGED3

Display all current macro definitions.

PAINT^RGEDS

3. Application Programming Interface

PAINT^RGEDS

Refresh the entire display.

\$\$PARSE^RGEDH()

Return the keyword located at the current edit position. This is implemented as an argumentless extrinsic function.

Example: S X=\$\$PARSE^RGEDH()

On return, X contains the keyword at the current edit position.

PASTE^RGED6(RGBUF, RGITER)

Insert RGITER copies of the contents of the source buffer named in RGBUF into the current edit position of the active buffer. The source buffer may not be the active buffer.

RGBUF (optional)	The name of the buffer whose contents is to be inserted in the active buffer. This should be a single character in the range of "A"-"Z", "0"-"9" or "@". A null value specifies the main buffer. If not specified, the editor prompts for a buffer name.
RGITER (optional)	The number of copies to be inserted. If not specified, defaults to one.

Example: D PASTE^RGED6("A",5)

Inserts 5 copies of the A buffer at the current edit position of the active buffer.

Example: D PASTE^RGED6("@")

Inserts one copy of the delete buffer at the current edit position of the active buffer. This is identical to the undelete operation.

Example: D PASTE^RGED6()

Inserts one copy of a user-specified buffer at the current edit position of the active buffer.

PGDN^RGED2(RGITER)

Move down RGITER pages. The column position remains unchanged. A page is defined as 2 less than the maximum number of text lines that can be displayed in the edit window. If an attempt is made to move beyond the end of the buffer, the editor signals an error and moves to the line following the last buffer line.

RGITER (optional)	The number of pages to traverse. If not specified, defaults to one.
----------------------	---

Example: D PGDN^RGED2(10)

Moves down 10 pages.

PGUP^RGED2(RGITER)

Move up RGITER pages. The column position remains unchanged. A page is defined as 2 less than the maximum number of text lines that can be displayed in the edit window. If an attempt is made to move beyond the beginning of the buffer, the editor signals an error and moves to the first line.

RGITER (optional)	The number of pages to traverse. If not specified, defaults to one.
----------------------	---

Example: D PGUP^RGED2()

Moves up one page.

POS^RGED2

Position the display cursor at the current edit position. The editor insures that the display cursor is properly positioned at the start of each command cycle. However, individual subroutines may alter and not restore this position. Therefore, if you must be certain that the cursor is properly positioned after a call to a subroutine, call *POS^RGED2*.

PRINT^RGED3

Output the active buffer to the selected device. This is accomplished through the %ZIS device handler and supports queuing of output.

`$$PRMPT^RGED2(RGPRMPT,RGLEN,RGFLG)`

Display a message on the message line. Optionally the editor may prompt for input from the user by invoking the line editor (^RGMSCEDT)¹. This operation is implemented as an extrinsic function whose return value is the input requested from the user (or null if no input was requested).

RGPRMPT	If numeric, its absolute value is the index of the message in the <i>EXTENSIBLE EDITOR MESSAGE</i> file. If zero or null, causes the default information text to be displayed (see the <i>INFORMATION TEXT</i> field description in section 1.1). If a negative number, indicates that this is an error message and causes the editor to sound the bell and clear the iteration counter. Otherwise, is the actual text of the message to be displayed. In any case, the text may contain “ ”-delimited replaceable parameters.
RGLEN (optional)	If nonzero, the editor awaits user input of up to RGLEN characters. If not specified, defaults to the screen width less the length of the message (prompt). If zero, the editor does not request user input (i.e., it only displays the message). How the editor behaves once the specified input length is reached depends upon the setting of the RGFLG argument (see the "I", "H", and "T" flags).
RGFLG (optional)	Modifies the default behavior of the user input operation. The following characters are valid: I = Maintain the cursor at the current edit position instead of positioning it following the input prompt. The "E" and "T" options are automatically invoked as well. E = Suppress echoing typed characters.

¹Detailed documentation on the line editor may be found in the Software Development Utilities Technical Manual which is available through the Department of Medical Informatics at the Indianapolis VAMC.

RGFLG (continued)	<p>H = Enable horizontal scrolling of input longer than the RGLen argument. By default, input is limited to the length specified by the RGLen argument.</p> <p>I = Suppress input time-out, ignoring the setting of the DTIME variable.</p> <p>L = Force all input to lower case.</p> <p>O = Begin input in overwrite mode. In this mode, typed characters replace those at the current insertion point. By default, insert mode is active causing existing characters to be displaced to the right as new text is entered. The user may toggle this mode at any time by typing <control>A.</p> <p>T = Auto-terminate input when the maximum input length as specified by the RGLen argument has been reached. This is the default mode when RGLen equals 1. Otherwise, the default mode terminates input only when the RETURN or ENTER key is pressed.</p> <p>U = Force all input to upper case.</p> <p>X = Suppress auto-erase. By default, any initial input text displayed by virtue of the RGDATA argument is automatically erased as the user begins typing unless one of the edit control keys (e.g., the arrow keys) is first typed in which case the initial text is preserved and may be modified just as if it had been entered by the user. This option suppresses this default behavior.</p>
RGVALID (optional)	If specified, must be a string of all characters that may be entered at the prompt. Any attempt to enter a character not contained within this argument will be ignored.
RGDATA (optional)	If specified, the input field is filled with the contents of the argument. Otherwise, the input field is initially null.

Example: S X=\$\$PRMPT^RGED2(2)

Prompts the user for input using message #2. On return, X contains the value entered by the user.

Example: S X=\$\$PRMPT^RGED2("Enter an integer value:",5,"","0123456789","0")

Prompts the user for up to 5 characters using the prompt "Enter an integer value:". The user is limited to entering only numeric digits. An initial value of 0 is displayed. On return, X contains the value entered.

Example: S X=\$\$PRMPT^RGED2(-5,0)

Displays message #5 without requesting input. Because the message number is negative, the editor sounds the bell and clears the iteration counter. On return, X will be null.

Example: S X=\$\$PRMPT^RGED2("Replace text?",1,1)

Prompts the user for single character input using the prompt "Replace text?". Because the third argument is nonzero, the editor does not reposition the display cursor before requesting input. On return, X will contain the single character typed.

QUIT^RGED0

Exit the editor. Actually does not exit until the beginning of the next command fetch cycle. If changes have been made to the main buffer, the user is given the option to save the changes or abort the operation.

REPLACE^RGEDF(RGITER)

Replace RGITER occurrences of a user-specified search string with a user-specified substitution string. See *SRCRPL^RGEDF* for more information on text substitution.

RGITER (optional)	Determines the number of occurrences that are to be substituted. If 0 or not defined, the editor searches for the next occurrence and requests confirmation before proceeding. If nonzero, the editor replaces the next (or previous) n occurrences of the search string with the substitution string.
----------------------	--

Example: D REPLACE^RGEDF(5)

Replaces the next (or previous) 5 occurrences of a user-specified search string with a user-specified substitution string.

RESTST^RGED1

Restore the current user's most recently saved state variable and tab stop settings. The status line is updated automatically to reflect the new settings. Any executable code associated with a restored state variable (through the *EXECUTE CODE* subfield of the *STATE VARIABLE* multiple) is also invoked at this time.

RIGHT^RGED2(RGITER)

Move the current edit position right RGITER character positions. A line break counts as a single character and traversing one results in the edit position moving to the beginning of the line that follows. If an attempt is made to move past the end of the buffer, the editor signals an error and moves the edit position to the last character position of the line following the last buffer line.

RGITER (optional)	The number of character positions to move. If not specified, defaults to one.
----------------------	---

Example: D RIGHT^RGED2(20)

Moves right 20 character positions.

RM^RGED1(RGC)

Set the right margin setting to the column position given by RGC. The ruler, if active, is automatically updated. The right margin cannot be within 5 columns to the right of the left margin. The editor signals an error if this occurs.

RGC (optional)	The column position that is to become the new right margin setting. If not specified, defaults to the current column position.
-------------------	--

Example: D RM^RGED1()

Sets the right margin to the current column position.

Example: D RM^RGED1(80)

Sets the right margin to column 80.

RULER^RGEDS

Display the ruler line, if enabled. The setting of the ruler line state variable (#6) determines how this subroutine operates.

\$\$RV^RGEDS(RGROW,RGCOL)

Set the video attribute for the buffer position given by RGROW and RGCOL. If the specified position is part of a text selection, sets the reverse video attribute. Otherwise, clears the reverse video attribute. Implemented as an extrinsic function, the return value is always null. The values of \$X and \$Y are preserved.

RGROW	The number of the line to evaluate. The first line is designated 1.
RGCOL	The column position to evaluate. The first column is designated 1.

Example: W \$\$RV^RGEDS(5,1),"X"

Sets the video attribute for the first column in line #5 and then displays "X" using that attribute.

SAVE^RGED0

Save the current contents of the main buffer to the word-processing field specified on invocation of the editor. If the main buffer is not active, it is activated before saving. This subroutine ignores the initial setting of the RGCHG variable, but does set it to zero on exit.

SAVEST^RGED1

Save the current settings of the user's state variables and tab stops. If the user is not identified (i.e., no DUZ), an error message is displayed and no action is taken. User settings are stored in the ^RGED global indexed by the internal configuration identifier and the DUZ. Therefore, each user may have their own default settings for each configuration used.

Note that the *LOAD USER DEFAULTS* subfield in the *EXTENSIBLE EDITOR* file controls whether or not settings are actually saved and subsequently restored for a given state variable. For some state variables, the ability to set user defaults may not be desirable. An example is the state variable that indicates a text selection is active. To save and subsequently restore the value of this variable might be confusing to the user.

SCRBTM^RGED2

Move the current edit position to the last line in the edit window. The column position remains unchanged.

SCRLT^RGED2

Move the current edit position to the leftmost column in the edit window. The line position remains unchanged.

SCRN`RGEDS(RGX, RGY)

Initialize display characteristics using the X- and Y-coordinate extents provided. This subroutine sets the display width (80 or 132 column), scroll regions, and other characteristics needed for proper display. It also initializes the display extent variables RGX1, RGX2, RGY1, and RGY2.

RGX (optional)	This specifies the display width. Legal values are 80 or 132. If not specified, toggles the current setting of the display width state variable (#11) and uses that value.
RGY (optional)	This specifies the number of available display rows. Should not exceed the number of rows provided by the display device, but can be less if desired. If not specified, uses the value of the display height state variable (#12).

Example: D SCRN`RGEDS()

Toggles the current display width setting (80 becomes 132 and vice-versa) and sets the number of display rows equal to the value of the display height state variable.

Example: D SCRN`RGEDS(RGST(11),RGST(12))

Sets the display characteristics according to the values in the display width and height state variables.

SCRRT^RGED2

Move the current edit position to the rightmost column in the edit window. The line position remains unchanged.

SCRTOP^RGED2

Move the current edit position to the first line in the edit window. The column position remains unchanged.

SEARCH^RGEDF(RGITER)

Search for the RGITER occurrence of a text string. The editor will prompt for the search string. A null entry uses the previously entered value (stored in RGSRCH). The editor then searches for the nth occurrence (i.e., the value of RGITER) of the string starting at the current edit position and proceeding in the direction determined by the value of the search direction state variable (#3). Case sensitivity of the search is determined by the setting of the case sensitivity state variable (#4). The editor positions the current edit position at the beginning of the located text, if any.

RGITER (optional)	The text string occurrence to be located. If not specified, defaults to the first.
----------------------	--

Example: D SEARCH^RGEDF(5)

Searches for the 5th occurrence of a user-specified text string. If the search fails, the editor signals an error and sets the edit position to the beginning of the last occurrence found or, if none were found, leaves the edit position unchanged.

SELECT^RGED5

Activate or deactivate text selection, depending on the current state of the text selection state variable (#14).

If text selection is active when this subroutine is called, the selection is canceled. The text selection state variable and anchor variables are cleared and selected text is unhighlighted.

If text selection is not active when this subroutine is called, the text selection variable is set to one and the anchor variables (RGSR1, RGSC1, RGSR2, and RGSC2) are set to the current edit position. Subsequent repositioning of the edit position causes the editor to highlight in reverse video all text between the anchor and the new position.

To determine if a text selection is active, you may test the value of the text selection state variable or the anchor variable RGSR1. Both will be zero if no selection is active.

SETTAB^RGED1(RGINT)

Sets a tab stop every RGINT columns. If RGINT is not specified, prompts for a value. Any previously set tab stops are lost.

SHOW^RGEDS

Refresh the edit window of the display.

SHST^RGEDS(RGST,RGTGL)

Display and/or modify the status of the state variable RGST. State variables should be modified only by calling this subroutine. If state labels are associated with the variable, the editor automatically displays the updated setting on the status line. If executable code is associated with the variable, it is called after updating its value.

See the section on state variables for additional information.

RGST	The number of the state variable whose status is to be viewed and/or modified.
RGTGL (optional)	If greater than or equal to 0, the state variable is changed to that value. If less than 0, the state variable is incremented by one. If the result exceeds the maximum allowable value, it is set to 0. If this argument is not specified, the state variable is not modified, but its current status is updated on the status line (if it has associated state labels) and any associated code is executed.

Example: D SHST^RGEDS(5)

Displays the status of state variable #5. If any executable code is associated with the variable, it is invoked.

Example: D SHST^RGEDS(583001,80)

Sets the value of state variable #583001 to 80.

Example: D SHST^RGEDS(10,-1)

Toggles the value of the word wrap state variable (#10). Because this variable has a maximum value of 1, this call effectively toggles its value between 0 and 1 (i.e., if it is already 1, incrementing it exceeds the maximum allowable value and the subroutine sets it to 0).

SIZE^RGEDX2

Compute the size of the M language routine currently being edited. The routine editor REDIT loads selected routines into a temporary word processing field and invokes the PROG configuration of the editor. The start of each routine is marked by a line containing the routine name prefixed by an asterisk. The editor uses this routine separator to determine the boundaries of the currently edited routine, scans the buffer to compute the size in bytes, and then displays the result on the message line.

This is an exported extension for the PROG configuration of the editor.

SLINE^RGEDS(RGLNM)

Display line # RGLNM at the current cursor position. You are responsible for insuring proper display cursor positioning before calling the subroutine.

The editor automatically clips the beginning and end of the line according to the current view of the active buffer. If the displayed line is part of an active text selection, that portion common to the selection will be automatically highlighted in reverse video. If the text contains undisplayable characters, these are automatically mapped to displayable surrogates (space for tab character, \$ for escape character, and ~ for all others).

This subroutine is useful if only a few lines need to have their display images updated. For larger display updates, the *SHOW^RGEDS* subroutine may be preferred.

RGLNM	The number of the line to be displayed.
-------	---

Example: W \$\$^RGXY(RGX1, RGROW-RGTOP+RGY1)
 D SLINE^RGEDS(RGROW)

Positions the cursor and then displays the current line.

SRCRPL^RGEDF(RGS, RGITER, RGRF, RGR)

Perform a text search and optionally a text substitution operation. The editor performs a text search from the current edit position and proceeding in the direction determined by the setting of the search direction state variable (#3). Search case sensitivity is controlled by the setting of the case sensitivity state variable (#4). When text substitution is selected, the located search string is replaced by the value of the replacement string.

RGS (optional)	The text string to be located. If unspecified or null, the editor prompts for the string.
RGITER (optional)	If not a text substitution, defines the nth occurrence of the search string to locate. If a text substitution and this value is nonzero, defines the number of occurrences that are to be substituted without confirmation. If zero or not specified, the editor performs substitutions according to the setting of RGRF.
RGRF (optional)	If not defined, the subroutine performs a search only function. If defined, determines how the editor handles a pending substitution. May be one of 5 values: Y=perform the substitution and stop, N=do not perform the substitution and stop, C=perform the substitution and continue searching, S=do not perform (skip) the substitution and continue searching, and A=perform this and all subsequent substitutions without further prompting. If null, the editor will prompt for its value.
RGR (optional)	The string that is to be substituted for the located text. If not specified and RGRF is defined, the editor will prompt for its value.

Example: D SRCRPL^RGEDF()

The editor prompts for a search string and locates its next (or previous) occurrence.

Example: D SRCRPL^RGEDF("TEXT",5)

The editor search for the 5th occurrence of "TEXT" following (or preceding) the current position.

Example: D SRCRPL^RGEDF("TEXT",5,"","NEW")

The editor replaces the next (or previous) 5 occurrences of "TEXT" with the "NEW".

Example: D SRCRPL^RGEDF("TEXT",0,"","NEW")

The editor locates the next (or previous) occurrence of "TEXT", then prompts for confirmation before performing the substitution.

Example: D SRCRPL^RGEDF("TEXT",0,"A","")

The editor replaces all occurrences of "TEXT" that follow (or precede) the current position with the null string without requesting confirmation. This effectively deletes all located occurrences.

STATUS^RGEDS

Display the status and title lines.

SWITCH`RGED5(RGBUF)

Make the buffer named in RGBUF the active buffer. This operation saves all context-specific variables for the current buffer and loads those for the new buffer. The display is updated accordingly. You may not make the delete buffer the active buffer.

RGBUF (optional)	The name of the buffer that is to become the active buffer. This should be a single character in the range "A"-"Z" or "0"-"9" or null to specify the main buffer. If not specified, the editor prompts for a buffer name.
---------------------	---

Example: D SWITCH`RGED5("X")

Make the X buffer the active buffer.

TAB^RGED2(RGITER)

Move right RGITER tab stops. If there are no more tab stops, no action is taken.

If the tab mode toggle (state variable #16) is set to soft (0) only the edit position changes. If set to hard (1) existing text to the right of the current edit position shifts to the right to align with the tab stop before the edit position is changed. Note that vacated character positions are filled with spaces, not tab characters, and that subsequent changes in tab settings do not affect previously modified text.

RGITER (optional)	The number of tab stops to traverse. If not specified, defaults to one.
----------------------	---

Example: D TAB^RGED2()

Moves right one tab stop.

TAB^RGEDH

Moves to the next hypertext link in a help frame. Valid only when a help-style configuration is active.

TGLTAB^RGED1(RGC, RGSET)

Toggle/set the setting of the tab stop at the specified column position. The ruler, if active, automatically displays the updated settings.

RGC (optional)	The column position of the tab to be modified. If not specified or is 0 or null, defaults to the current column position.
RGSET (optional)	If not specified, the current state of the tab stop is toggled. If 0, the tab stop is cleared. Otherwise, the tab stop is set.

Example: D TGLTAB^RGED1()

Toggles the state of the tab stop at the current column position.

Example: D TGLTAB^RGED1(5)

Toggles the state of the tab stop at column 5.

Example: D TGLTAB^RGED1(20,1)

Sets a tab stop at column 20.

TOP^RGED2

Move the current edit position to the beginning of the buffer. The column remains unchanged.

UNALIAS`RGED4

Remove the current configuration as an alias. If the current configuration is not invoked as an alias for another, this subroutine has no effect. Otherwise, the current configuration will no longer be loaded as an alias for the original configuration on subsequent invocations of the editor.

See also *ALIAS`RGED4* for more information on aliasing.

UNDLIN^RGED4(RGITER)

Insert text deleted on last line or block deletion operation at the current edit position. Whenever a line deletion operation (partial or complete) or a block deletion operation (i.e., a deletion on a selection of text) occurs, the editor stores the deleted text in the delete or "@" buffer. This text remains recoverable until the next deletion occurs. Calling this subroutine is precisely equivalent to pasting from the delete buffer.

RGITER (optional)	The number of copies of deleted text to be inserted. If not specified, defaults to one.
----------------------	---

Example: D UNDLIN^RGED4(5)

Deletes the current line and the four lines that follow.

UNDO^RGED2

Restores the current line to its previous state.

The editor does not commit changes made to the current line until the edit position moves to another line or a function that operates directly on the text buffer is invoked. The undo function restores the current line to its last committed state.

UP^RGED2(RGITER)

Move the current edit position up RGITER lines. If an attempt is made to move beyond the beginning of the buffer, the editor signals an error and moves to the first line.

RGITER (optional)	The number of lines to move. If not specified, defaults to 1.
----------------------	---

Example: D UP^RGED2(5)

Moves up 5 lines.

UPDATE^RGED2

Update changes made to the current line. For efficiency, the editor buffers changes made to the current line in the variable RGLN. Whenever the current line changes or a subroutine needs to directly access the buffer, the editor writes the current value of RGLN back to the buffer. Therefore, you should make changes directly to RGLN when modifying the current line and should make a call to *UPDATE^RGED2* before directly accessing the edit buffer.

WLEFT^RGED2(RGITER)

Move the current edit position RGITER words left. A word is defined as contiguous alphanumeric text.

RGITER (optional)	The number of words to traverse. If not specified, defaults to one.
----------------------	---

Example: D WLEFT^RGED2()

Moves 1 word left.

3. Application Programming Interface

WRAP^RGEDI

Perform word wrap on the current line. Word wrap occurs if the line exceeds the right margin setting and regardless of the setting of the word wrap state variable. The edit position is automatically adjusted if necessary to maintain the same relative text position.

WRIGHT^RGED2(RGITER)

Move the current edit position right RGITER words. A word is defined as contiguous alphanumeric text.

RGITER (optional)	The number of words to traverse. If not specified, defaults to one.
----------------------	---

Example: D WRIGHT^RGED2(5)

Moves 5 words to the right.

3. Application Programming Interface

4. Utilities

Six utility programs are distributed with the Extensible Editor. Two are configuration aids and four are editing aids.

4.1 *RGED*

This program is the main entry point for the Extensible Editor. Because it may also be invoked recursively from within the editor environment, it is discussed in detail in section 3.5 as part of the editor API. This discussion also applies to invoking the editor from a separate application.

4.2 *RGEDCMP*

This program compiles configuration key map tables. Any time a configuration's key map table or key macro table is altered, the configuration must be recompiled. When invoked, the routine prompts for a configuration name. Specify the name of the configuration to be compiled, or **ALL** to compile all existing configurations. This utility is also described in section 2.5.

4.3 *RGEDFIL*

This utility permits editing of ASCII text files. There are two entry points.

The first entry point prompts for a filename and then opens the file in read/write mode.

Example: D ^RGEDFIL
 Filename to edit: **TEST.FIL**

The second entry point has one required and one optional argument. The first argument is the filename and should include any navigational information required to locate the file. The second argument is the name of the output file. If this argument is not specified, changes are saved to the original file. If this argument is null, the input file is opened in read-only mode.

Example: D ENTRY^RGEDFIL("FILE.TXT")

Loads the file "FILE.TXT" into the editor using the DEFAULT configuration. Changes will be saved back to the same file.

Example: D ENTRY^RGEDFIL("FILE.TXT","FILE2.TXT")

Loads the file "FILE.TXT" into the editor using the DEFAULT configuration. Changes will be saved to the file "FILE2.TXT". If this file already exists, its previous contents will be lost.

Example: D ENTRY^RGEDFIL("FILE.TXT","")

Loads the file "FILE.TXT" into the editor using the BROWSE configuration. The file may be viewed, but not modified.

4.4 REDIT

This utility permits the editing of M language routines. When invoked, you will be presented with a routine selection prompt. Select all routines you wish to edit. A null entry terminates the selection and invokes PROG configuration of the editor. Routines are presented in alphabetical order. Each begins with a line containing an asterisk followed immediately by the routine name. ***This line should never be modified or deleted.***

When the edited routines are saved, the editor places a time and date stamp on the second line of the routine in the sixth ";"-delimited piece along with the initials of the user. The user's initials are extracted from the *NEW PERSON* file. If the user is not identified, or their initials cannot be located, the routine prompts for the user's initials before saving the routines. If the routine being edited does not comply with the SACC convention for routine headers, the stamp is not applied.

Another feature of the routine editor is the provision for routine templates. Routine templates allow the user to create often used sections of code that can be used as shells for creating new routines. Since templates can contain imbedded replaceable parameters, one can actually create dynamic templates that automatically supply essential information when loaded.

A routine template is created like any other M routine. What distinguishes a template from a routine is the presence of a special tag on the first line. For the routine editor to recognize a template as such, the first line of the template must contain the tag "%ROUTINE TEMPLATE%" as the second piece of a semicolon-delimited string. The third piece of the string must contain the character or characters to be used to delimit replaceable parameters. If this third piece is null, the editor assumes there are no replaceable parameters within the template. To illustrate, consider the routine template shown below:

```

; %ROUTINE TEMPLATE%;z;
zRGZz ;CAIRO/DKM - z$$GET^RGEDIT("Brief description")z ; z$$^RGCVTDT($H)z
;;1.0;z$$GET^RGEDIT("Package")z;;z$$^RGCVTDT($H)z
;=====
===

```

When the editor attempts to load this routine, the presence of the tag on the first line signals that this is a routine template. The editor signals this fact to the user and prompts for a new routine name under which the edited code will be saved. The editor will not permit the selection of an existing routine name. Note that the third piece of the first line contains a lowercase z. This tells the editor to use that character to delimit replaceable parameters. The second line of the routine template begins with "zRGZz". This indicates that RGZ is a replaceable parameter (which will always contain the new routine name as entered by the user). When the editor loads the template, the "zRGZz" is replaced by the new routine name. The loaded template might appear something like this:

```
RGTEST ;CAIRO/DKM - Sample routine ; 19-Sep-96 11:00
;;1.0;Test;;19-Sep-96 11:00
```

```
;=====
```

Once loaded, the template-generated routine can be edited like any other routine and will be saved under the routine name selected by the user (in this case "RTN"). Note that if the user wishes to edit the template itself, entering a "@" when prompted for the new routine name will cause the editor to load the template in its raw form and to be subsequently saved under its original name.

In the example above, note the use of the extrinsic function \$\$GET^RGEDIT. When a reference to this function is included as a replaceable parameter in the template, it results in the user being prompted for the data to insert. The first argument is the prompt to be used, the second is an optional default value. In this way, templates can dynamically acquire input from the user.

Another entry point worthy of mention in RGEDIT is called SEARCH. If RGEDIT is invoked at this entry point, it enables the programmer to select a range of routines and then specify one or more search strings. Only the subset of selected routines that contain at least one of the search strings will be loaded into the editor.

4.5 *RGEDKD*

This utility facilitates the creation of keystroke macro definitions. It is described in detail in section.

5. Routines

PRE^RGEDINIT	Pre-initialization
POST^RGEDINIT	Post-initializarion
RGED	Extensible Editor
RGED0	Continuation of RGED
RGED1	Continuation of RGED
RGED2	Continuation of RGED
RGED3	Continuation of RGED
RGED4	Continuation of RGED
RGED5	Continuation of RGED
RGED6	Continuation of RGED
RGEDBRS	Text file browser
RGEDCMP	Initialize key map table
RGEDF	Continuation of RGED
RGEDFIL	CIRN/DKM - Text file editor
RGEDH	Continuation of RGED
RGEDINIT	Extensible Editor Inits
RGEDIT	M routine editor
RGEDKD	Create keystroke macros
RGEDS	Continuation of RGED
RGEDTMPL	Routine template
RGEDX1	Maintenance utilities
RGEDX2	Programmer extensions
RGEDX3	Spell checking extension
RGEDX4	Help frame editor

6. External Relations

DBIA's for which Extensible Editor is Custodian

2793 NAME: CIRN CALLS TO RGED2
 CUSTODIAL PACKAGE: EXTENSIBLE EDITOR
SUBSCRIBING PACKAGE: CLINICAL INFO RESOURCE NETWORK
 USAGE: Private APPROVED: APPROVED
 STATUS: Active EXPIRES:
 DURATION: Till Otherwise Agr VERSION:
 FILE: ROOT:
DESCRIPTION: TYPE: Routine
 CIRN ROUTINE DBIA TO CALL RGED2

ROUTINE: RGED2
COMPONENT: PRMPT
VARIABLES:

RGPRMPT Input

If numeric, its absolute value is the index of the message in the EXTENSIBLE EDITOR MESSAGE file. If zero or null, causes the default information text to be displayed (see the INFORMATION TEXT field description in section 1.1). If a negative number, indicates that this is an error message and causes the editor to sound the bell and clear the iteration counter. Otherwise, is the actual text of the message to be displayed. In any case, the text may contain "|" -delimited replaceable parameters.

RGLLEN Input

If nonzero, the editor awaits user input of up to RGLLEN characters. If not specified, defaults to the screen width less the length of the message (prompt). If zero, the editor does not request user input (i.e., it only displays the message). How the editor behaves once the specified input length is reached depends upon the setting of the RGFLG argument (see the "I", "H", and "T" flags).

COMPONENT: MOVETO

RGR Input

The line number to move to.

RGC Input

6. External Relations

The column position to move to. Move to a new buffer row and/or column

COMPONENT: UPDATE

Update changes made to the current line. For efficiency, the editor buffers changes made to the current line in the variable RGLN. Whenever the current line changes or a subroutine needs to directly access the buffer, the editor writes the current value of RGLN back to the buffer. Therefore, you should make changes directly to RGLN when modifying the current line and should make a call to UPDATERGED2 before directly accessing the edit buffer.

2794 NAME: CIRN AGREEMENT WITH EXTENSIBLE EDIT
 CALL TO RGEDS
 CUSTODIAL PACKAGE: EXTENSIBLE EDITOR
SUBSCRIBING PACKAGE: CLINICAL INFO RESOURCE NETWORK
 USAGE: Private APPROVED:
 STATUS: Pending EXPIRES:
 DURATION: Till Otherwise Agr VERSION:
 FILE: ROOT:
DESCRIPTION: TYPE: Routine

Private Routine DBIA for call to PAINT^RGEDS

ROUTINE: RGEDS
COMPONENT: PAINT

Refresh the entire display.

2797 NAME: CIRN AGREEMENT WITH EXTENSIBLE EDITOR FOR
 CALLS TO RGED
 CUSTODIAL PACKAGE: EXTENSIBLE EDITOR
SUBSCRIBING PACKAGE: CLINICAL INFO RESOURCE NETWORK
 USAGE: Private APPROVED: APPROVED
 STATUS: Active EXPIRES:
 DURATION: Till Otherwise Agr VERSION:

FILE: ROOT:
DESCRIPTION: TYPE: Routine

CIRN AGREEMENT WITH EXTENSIBLE EDITOR FOR CALLS TO RGED

ROUTINE: RGED
COMPONENT: RGED
VARIABLES:

RGDIC Input

This is the global root of the word-processing field being edited. The contents of the word- processing field are loaded into the main buffer during editor initialization.

RGTTL Input

The title of the editing session, displayed in the upper left-hand corner of the screen. Replaceable parameters are allowed.

RGOBJ Input

The name of the object being edited, displayed in the upper right-hand corner of the screen. Replaceable parameters are allowed.

RGCF Input

The name of the configuration that is to be invoked. This is the name of the .01 field in the Extensible Editor file. If not specified, the editor uses the DEFAULT configuration.

RGPID Input

This is the instance identifier for the editor. Normally, this parameter is not passed and defaults to the value of \$JOB. If the editor is invoked recursively (i.e., from within the editor itself), this parameter should be the current value of RGPID. The editor adds .0001 to this value to maintain a unique context across all instances. This is the entry point for invoking the Extensible Editor. It may be invoked outside the editor environment or recursively from within the editor environment. Only the first argument is required.

.....

7. Exported Options

RGED PURGE – This is a schedulable option that purges terminated user's settings from the ^RGED(global. The recommended frequency for running this option is once per month.

8. Archiving and Purging

RGED PURGE – This is a schedulable option that purges terminated user's settings from the ^RGED(global. The recommended frequency for running this option is once per month.

9. Internal Relations

All routines, files and options within the CIRN-PD software can function independently.

- 7. Exported Options
- 8. Archiving and Purging
- 9. Internal Relations

10 . Security

Files with Security Access

FILE #	NAME	DD ACCESS	RD ACCES S	WR ACCES S	DEL ACCES S	LAYGO ACCESS
996	EXTENSIBLE EDITOR					
996.2	EXTENSIBLE EDITOR SPELLCHECKER					

11. Package Wide Variables

The following variables are defined by the editor. All fall under the RG namespace which is the reserved namespace for this development site. Unless explicitly defined to be modifiable, they should be treated as read-only. For those that may be modified directly, close attention should be paid to any constraints that may accompany their modification. Failure to do so may cause erratic or otherwise undesirable behavior.

RGBEL	Contains the BELL character or is null if the bell has been disabled.
RGBMC	Column position of the current bookmark, or 0 if none defined.
RGBMR	Row position of the current bookmark, or 0 if none defined.
RGBUF	Internal identifier of the buffer currently active. Identifiers are of the format "RGEDx" where x is null if it is the main buffer or the letter of the active buffer otherwise.
RGCF	The internal identifier of the active configuration. This is actually the record number within the <i>EXTENSIBLE EDITOR</i> file.
RGCF2	If a configuration has been aliased as another, this contains the internal identifier of the original configuration, otherwise it will be 0.
RGCH	The string of unprocessed keystrokes.
RGCH1	Last character received from keyboard buffer.
RGCHG	Set to a nonzero value whenever the current buffer is modified. This indicates to the editor that a modification has taken place. You must set this variable to 1 if you modify the active buffer outside a standard API subroutine call. Failure to do so or setting it to 0 may result in changes being lost when the editor exits.
RGCLR	Contains the VT100 command sequence to clear the screen.
RGCOL	Column position of the current edit position. The first column is numbered 1.
RGCTL1	A string containing non-displayable ASCII characters.
RGCTL2	A string containing the display equivalents of the characters in RGCTL1.
RGDIC	This is the global root of the word-processing field being edited. This is one of the formal parameters passed on invocation of the editor.

RGEOL	Contains the VT100 command sequence to clear to the end of the line.
RGEOS	Contains the VT100 command sequence to clear to the end of the screen.
RGESC	Contains the escape character, ASCII 27.
RGFIN	If this variable exists, the editor will exit upon return from the subroutine that created it. You may set this variable, but do so with care. The editor does not save any changes, but exits immediately upon return.
RGHION	The VT100 escape sequence for activating the highlight attribute.
RGHLP	This contains the text of the default information message as defined in the configuration's <i>INFORMATION TEXT</i> field. This is the message displayed on the editor's message line when no other message is active. You may modify this if you wish, but its length should not exceed 50 bytes. You may include replaceable parameters.
RGITER	This is the value of the current iteration count. If your routine supports multiple iteration, it may use this variable and decrement it with each iteration. The iteration count is reset to 0 following execution of the first command after it is set.
RGITER2	This is the pending iteration count. The ITER^RGED1 subroutine sets this variable, which becomes the active iteration count (RGITER) on the next command fetch cycle.
RGLC	This is the line count for the active buffer.
RGLFT	This is the character offset of the leftmost character displayed. Because the editor supports horizontal text scrolling, the leftmost character may not be the first character in a line.
RGLN	This is the text of the line currently being edited. You may modify this as needed. The editor automatically writes changes back to the buffer whenever the current line changes or when the UPDATE^RGED2 subroutine is called.
RGMAX	This is the maximum allowable line length. It is set according to the value specified in the configuration's <i>MAXIMUM LINE LENGTH</i> field.
RGMSG	If nonzero, the editor will redisplay the default information message following execution of the next command. Normally, you should not need to modify this variable as the editor PRMPT^RGED2 subroutine sets it when a message is displayed.
RGNORM	The VT100 escape sequence that turns off all special display attributes.

RGOBJ	The name of the object being edited. This is displayed in the top right-hand corner of the screen. It may contain replaceable parameters. It may be modified if desired. This is one of the formal parameters passed on invocation of the editor.
RGPID	This is the instance identifier of the current invocation of the editor. It is one of the formal parameters optionally passed to the editor. On first invocation, it will normally equate to the value of the \$JOB system variable + .0001.. If the editor is entered recursively, this parameter is automatically incremented by .0001 to maintain a unique context.
RGROW	The current row being edited. The first row is designated row 1.
RGRULER	The ruler line. Tab stops are marked by "T". Intervening space is denoted by "=".
RGRVON	The VT100 character sequence for activating reverse video.
RGSC1	The column position of the first anchor point for a text selection.
RGSC2	The column position of the second anchor point for a text selection.
RGSR1	The row position of the first anchor point for a text selection. Will always be zero if no selection is active.
RGSR2	The row position of the second anchor point for a text selection.
RGSRCH	This is the value of the last search string used.
RGST	This is the table of state variables. See section 3.3 for more details.
RGTBL	This is the identifier of the active key map table. It may be modified with caution. Setting it to a nonexistent identifier will cause the editor not to respond to any keystrokes.
RGTOP	This is the row number of the first text line displayed on the screen.
RGTTL	This is the text title displayed in the upper left-hand corner of the screen. It is one of the formal parameters passed to the editor on invocation. It can be modified if desired. It may contain replaceable parameters.
RGUNON	The VT100 escape sequence for activating the underline attribute.
RGVER	The version identifier for this configuration. This is the value of the <i>VERSION</i> field.
RGX	The X-coordinate value of the current edit position.

11. Package Wide Variables

RGX1	The X-coordinate value of the leftmost column of the text window. Should always be 1.
RGX2	The X-coordinate value of the last column of the text window. Should be 80 or 132 depending upon the current width setting.
RGY	The Y-coordinate value of the current edit position.
RGY1	The Y-coordinate value of the first row of the text window.
RGY2	The Y-coordinate value of the last row of the text window.

12. Glossary

ASCII – American Standard Code for Information Interchange. A series of 128 characters, including upper and lower case. Alpha characters, numbers, punctuation, special symbols, and control characters.

API – Application Programmer Interface. These are callable entry points within the software that programmers can use to do pre-defined computing activities.

PC – Personal Computer

Index

\$

\$\$PARSE^RGEDH	76
\$\$PRMPT^RGED2	11, 82, 83, 84
\$\$RV^RGEDS.....	91

^

^RGED.....	See RGED utility
^RGED global	11, 93
^TMP global.....	21

A

Active buffer state variable	19
ALIAS^RGED4.....	25, 114
Aliasing.....	7, 16, 25, 114, 127, 137
Application programming interface	15
Application variables	15
ASCII^RGEDX2	26
AUTOSAVE ON TIMEOUT? field	4

B

BACK^RGEDH.....	27
BDEL^RGED5	28, 29
Bell tone state variable.....	20
BELL^RGED0	30
BKTB^RGED2.....	31
BKTB^RGEDH.....	32
BMFND^RGED1	33, 34
BMSET^RGED1	33, 34
Bookmarks	15, 33, 34, 137
BREAK^RGED1	35
BROWSE configuration	3, 10, 124
BTM^RGED2.....	36
Buffers	21

C

CALC^RGEDX2.....	37
CHARACTER SEQUENCE field.....	5, 10, 11, 12
CLRTAB^RGED1	38
CMD^RGED0	11, 39
CMNT^RGEDX2.....	40
CONFIGURATION field.....	3, 9, 10
Configurations, compiling	11, 123
Configurations, creating.....	9
COPY^RGED5.....	41
CREATION DATE field	3

CSH^RGEDH.....	42
Cursor, positioning the display.....	24, 80
CUT^RGED5	43
Cutting, copying, and pasting text	41, 66, 77

D

DEFAULT configuration	3, 5, 9, 10, 13, 23, 123
DEFAULT EXECUTABLE CODE field	5
DEFAULT STATE field.....	6
DELBOL^RGED4	44
DELEOL^RGED4	45
DELETE^RGED0	46
DELLIN^RGED4	47
DESCRIPTION field.....	6
DIALOG file	7, 13, 14, 127
DISABLE TIMED READS? field.....	4
Display height state variable	20
Display width state variable	20
DOWN^RGED2	48

E

Edit position tracking state variable.....	19
END^RGED2	11, 49
EXECUTABLE CODE field.....	5, 10, 11
EXECUTE CODE field	6, 87
EXTENSIBLE EDITOR file	3
EXTENSIBLE EDITOR MESSAGE file.....	7, 8

F

FileMan.....	1, 9, 10, 11, 13, 60
FILTER field.....	6, 12, 13
FIND^RGEDH	50
FLUSH^RGEDH.....	51
FNEXT^RGEDF	52
FORMAT^RGED1	53, 63

G

GEN^RGEDH	11, 54
GLOBAL ROOT field	6, 12
GOTO^RGED1	55

H

HELP configuration	3, 9
HELP FRAME (CONTEXT-SENSITIVE) field	4, 42
HELP FRAME (GENERAL).....	54
HELP FRAME (GENERAL) field.....	4
Help frames	4, 27, 32, 42, 50, 54, 56, 59, 66, 68, 111
HELP^RGEDH	56
HLPMSG^RGEDS	57

HOME^RGED2.....	58
HOME^RGEDH.....	59
Hypertext help	1, 3, 9, 20, 32, 50, 66, 68, 111

I

Import filters.....	3, 6, 9, 12, 13, 60
IMPORT^RGED6	60, 61
INFORMATION TEXT field	4, 16, 57, 82, 138
Insert mode state variable.....	19
INSERT^RGED0	11, 62, 71, 127
Instance identifier.....	See RGPID. See RGPID
ITER^RGED1	17, 63, 138

J

JOIN^RGED1.....	64
JUSTIFY^RGED4	65

K

KEY MACRO field	5
Key mappings, compiling.....	11, 123
KEYMAP TABLE field.....	5
Keystroke macros, creating	10, 125
Keystroke macros, using	3, 5, 10, 11, 12, 71, 72, 74, 123, 125
KEYW^RGEDH.....	66

L

Left margin state variable.....	20
LEFT^RGED2.....	67
LINK^RGEDH.....	68
LM^RGED1	20, 69
LMAR^RGED2.....	70
LOAD USER DEFAULTS field	6, 93

M

Margin settings.....	20, 40, 62, 65, 69, 70, 89, 120
MAXIMUM LINE LENGTH field	4, 17, 64, 138
MDEF^RGED3	71, 72
MEXE^RGED3	71, 72
MOVETO^RGED2.....	21, 73
MSHOW^RGED3	74

N

NUMBER OF POSSIBLE STATES field.....	6
--------------------------------------	---

P

PAINT^RGEDS	21, 75
Parameters, replaceable.....	3, 4, 7, 13, 16, 17, 18, 23, 82, 124, 138, 139
PASTE^RGED6	77
PGDN^RGED2	78

PGUP^RGED2	79
POS^RGED2.....	80
POSTINITIALIZATION field	4
PRINT^RGED3.....	81
PROG configuration	3, 10, 26, 37, 40, 104, 124

Q

QUIT^RGED0.....	85, 127
-----------------	---------

R

REPLACE^RGEDF.....	86
RESTST^RGED1	87
RGBEL	15, 30, 137
RGBMC.....	15, 137
RGBMR.....	15, 137
RGBUF.....	15, 21, 28, 29, 41, 43, 77, 109, 137
RGCF	15, 23, 25, 137
RGCF2.....	16, 137
RGCH.....	16, 50, 62, 137
RGCH1	11, 16, 137
RGCHG	16, 92, 137
RGCLR.....	16, 137
RGCOL.....	16, 91, 137
RGCTL1	16, 137
RGCTL2	16, 137
RGDIC.....	12, 13, 16, 23, 60, 137
RGED utility	23, 123
RGEDCMP utility	123
RGEDFIL utility	123
RGEDIT utility.....	3, 104, 124
RGEDKD utility.....	10, 11, 125
RGEOL.....	16, 138
RGEOS	16, 138
RGESC	16, 138
RGFIN	16, 138
RGHLP	16, 138
RGITER...16, 17, 31, 46, 47, 48, 52, 53, 60, 62, 63, 65, 67, 72, 77, 78, 79, 86, 88, 99, 106, 110, 115, 117, 119, 121, 138	
RGITER2.....	17, 63, 138
RGLC	17, 21, 138
RGLFT	17, 138
RGLN	17, 21, 118, 138
RGMAX	17, 138
RGMSG.....	17, 138
RGOBJ	17, 23, 139
RGPID.....	17, 21, 23, 139
RGROW	17, 21, 91, 105, 139
RGRULER.....	17, 139
RGRVON	17, 139

RGSC1.....	17, 100, 139
RGSC2.....	17, 100, 139
RGSR1.....	18, 100, 139
RGSR2.....	18, 100, 139
RGSRCH.....	18, 99, 139
RGST.....	12, 18, 19, 96, 103, 139
RGTBL.....	18, 39, 139
RGTOP.....	18, 105, 139
RGTTL.....	18, 23, 139
RGVER.....	3, 13, 18, 139
RGX.....	18, 24, 96, 139
RGX1.....	18, 96, 105, 140
RGX2.....	18, 96, 140
RGY.....	18, 24, 96, 140
RGY1.....	18, 96, 105, 140
RGY2.....	18, 96, 140
Right margin state variable	20
RIGHT^RGED2	88
RM^RGED1	20, 89
Routine templates	124
RULER field	4, 90
Ruler state variable.....	20
RULER^RGEDS	90

S

SAVE^RGED0.....	92
SAVEST^RGED1	93
SCRBTM^RGED2	94
SCRLT^RGED2.....	95
SCRN^RGEDS.....	20, 96
SCRRT^RGED2.....	97
SCRTOP^RGED2	98
Search and replace operations	1, 18, 50, 52, 86, 99, 106, 139
Search case sensitivity state variable	19, 99, 106
Search direction state variable.....	19, 52, 99, 106
SEARCH^RGEDF.....	52, 99
SELECT^RGED5.....	20, 100
SETTAB^RGED1	101
SHOW^RGEDS	102, 105
SHST^RGEDS	6, 19, 20, 30, 103
SIZE^RGEDX2	104
SLINE^RGEDS.....	105
Soundex codes	8
Special text handling state variable.....	20
Spell checking.....	8
SRCRPL^RGEDF	86, 106, 107
STATE LABELS field.....	6
STATE TABLE field.....	5, 6, 12, 18
State variables	12, 18

STATUS^RGEDS	108
SWITCH^RGED5	19, 21, 109
T	
Tab mode state variable	20
TAB^RGED2	110
TAB^RGEDH	111
Text selection state variable	20
TGLTAB^RGED1	112
TOP^RGED2.....	113
U	
UNALIAS^RGED4	114
UNDLIN^RGED4	115
UNDO^RGED2	116
UP^RGED2	117
UPDATE^RGED2	17, 21, 118, 138
User-defined settings	6, 11, 12, 18, 87, 93
Utilities.....	123
V	
VERSION field	3
VT100-compatibility state variable	20
W	
WLEFT^RGED2	119
Word wrap	5, 6, 20, 53, 103, 120
Word wrap state variable	20, 103, 120
WRAP^RGED1	120
WRIGHT^RGED2.....	121

Acknowledgements

The Extensible Editor was designed and developed by Douglas K. Martin, MD, Director of the Center for Applied Informatics Research and Operations (CAIRO) at the Richard L. Roudebush VAMC in Indianapolis. This work was supported in part by a VA Health Services Research and Development grant (CDS 91-306).

This work would not have been possible without the invaluable technical assistance and feedback from Mr. M. Randall Cox and Mr. Phillip L. Salmon of the Richard L. Roudebush VAMC in Indianapolis.

Acknowledgements